Australian Government
Department of Defence
Defence Science and
Technology Organisation

On the Design of a Comprehensive Authorisation Framework for Service Oriented Architecture (SOA)

Sarath Indrakanti

Cyber and Electronic Warfare Division
Defence Science and Technology Organisation

DSTO-TN-1193

ABSTRACT

Service Oriented Architecture (SOA) has attracted considerable industry attention because of the benefits it offers such as allowing interoperability over a heterogeneous environment, amongst others. However, security is one of the main roadblocks for enterprises when it comes to the development and deployment of their SOAs. Although there are several SOA security standards available, there is as yet no standard available for SOA authorisation. In this report, we propose a comprehensive authorisation framework for SOA.

Published by

Cyber and Electronic Warfare Division
DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh South Australia 5111 Australia

Telephone: 1300 DEFENCE
Fax: (08) 7389 6567

AR-015-670
July 2013

APPROVED FOR PUBLIC RELEASE

On the Design of a Comprehensive Authorisation Framework for Service Oriented Architecture (SOA)

Executive Summary

Over recent years, Service Oriented Architecture (SOA) has attracted considerable industry attention because of the benefits it offers such as allowing interoperability over a heterogeneous environment, amongst others. SOA can be used to build new solutions leveraging services, to integrate existing applications, or to cleave apart existing applications. SOA provides many benefits, such as cost saving to organisations by increasing the speed of implementation of any application(s), and reducing the expenditure on integration technologies. However, security is one of the main roadblocks for enterprises when it comes to the development and deployment of their SOAs.

There are several security standards available to guide secure design, development and deployment of SOA in organisations. There have also been several research and development efforts in both industry and academia striving to provide security services for SOA. However, there is as yet no standard available for SOA authorisation. Having looked at the design requirements for SOA authorisation by means of an extensive literature survey in previous work, we developed an authorisation model for SOA.

In this report, we propose a comprehensive authorisation framework for SOA. The framework comprises the Web Services Authorisation Architecture (WSAA) and the Business Processes Authorisation Architecture (BPAA) — designed for the Web service and business process layers of SOA. We describe the architectural framework, the administration and runtime aspects of both our architectures and their components for secure authorisation of Web services and business processes as well as the support for the management of authorisation information. We also discuss the benefits of our authorisation framework for SOA.

Our understanding is that IBM's Defence Operations Platform (DOP)[1] has been adopted by Australian Defence to build SOA services. We believe there is scope in the future to take into consideration real Defence services and their access control requirements, and research potential integration points of our comprehensive SOA authorisation framework into the DOP.

[1] http://www-01.ibm.com/software/industry/defense-operations-platform/

This page is intentionally blank

Contents

Acronyms

Acronym	Full Name
AAD	Authorisation Administration Database
ARD	Authorisation Runtime Database
ACL	Access Control List
ACO	Authorisation Coordinator
ADC	Authorisation Decision Composer
APE	Authorisation Policy Evaluator
AS	Activation Service
ANS	Authentication Server
AZM	Authorisation Manager
AZS	Authorisation Server
BP	Business Process
BPAA	Business Process Authorisation Architecture
BPAD	Business Process Administration Database
BPEL	Business Process Execution Language
BPSM	Business Process Security Manager
C2	Command and Control
CCA	Certificate and Credential Authority
CP	Client Proxy
CRM	Credential Manager
DAC	Discretionary Access Control
DAS	Dynamic Attribute Service
IPsec	Internet Protocol Security
JAAS	Java Authentication and Authorisation Service
MAC	Mandatory Access Control
RBAC	Role Based Access Control
RCA	Regional Commander Agent
RS	Registration Service
SCA	Strategic Commander Agent
SM	Security Manager
SOA	Service Oriented Architecture
SSL	Secure Sockets Layer
TCA	Tactical Commander Agent
TLS	Transport Layer Security
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
URN	Uniform Resource Name
WAD	Web service Administration Database
WS	Web Service
WSAA	Web Service Authorisation Architecture
WS-BPEL	Web Services Business Process Execution Language
WSC	Web Service Collection
WSCM	Web Service Collection Manager
WSDL	Web Services Description Language
WSM	Web Service Manager
XML	eXtensible Markup Language

This page is intentionally blank

1. Introduction

In general, security for Service Oriented Architecture (SOA) is a broad and complex area covering a range of technologies. At present, there are several efforts underway that are striving to provide security services for SOA. A variety of existing technologies can contribute to this area such as TLS/SSL [1] and IPsec [2]. There are also related security functionalities such as XML Signature [3] and XML Encryption [4] and their natural extensions to integrate these security features into Web service technologies such as SOAP [5] and WSDL [6].

The WS-Security specification [7] describes enhancements to SOAP messaging to provide message integrity, confidentiality and authentication. The WS-Trust [8] language uses the secure messaging mechanisms of the WS-Security specification to define additional primitives and extensions for the issuance, exchange and validation of security tokens within different trust domains. While there is a large amount of work on general access control and more recently on distributed authorisation [9], research in the area of authorisation for Web services is still at an early stage. There is not yet a specification or a standard for Web services authorisation. There are attempts by different research groups [10-13] to define authorisation models for Service-Oriented Architecture (SOA). Currently most Web service based applications, having gone through the authentication process, make authorisation decisions using application specific access control functions that results in the practice of frequently re-inventing the wheel. This motivated us to have a closer look at authorisation challenges for SOA. Please note that the work discussed in this report borrows heavily from the author's PhD thesis [14].

2. SOA Authorisation Challenges

SOA comprises Web services and business workflows (also called business processes [15]) built using Web services. Figure 1 shows the layers comprising SOA along with their respective security services.

Web services may potentially expose business logic presented via a complex layered system. For example, a Web service could be a portal service rendered via a Web browser that shows information from various sources such as files, databases and RSS feeds. The same portal service may provide functionality written using new Web services designed from scratch. Alternatively, a Bank may expose a legacy application (written in Cobol) as a Web service. An authorisation service designed for the Web services layer of SOA needs to consider all these scenarios. It must be able to support multiple models of access control used by these types of services. The access control models used may be established ones such as Role Based Access Control (RBAC) [16] or perhaps even new ones designed for the use case in question.

Consider the Sales Order service shown in Figure 2. Each method of the Sales Order service performs one or more of these three operations — Web operation, Mail operation and Database operation. Each of these operations uses a different access control mechanism. The Web operations may be authorised using, for instance, the Java Authentication and Authorisation Service (JAAS) [17], the Mail operations using a simple access control list (ACL), and the Database operations using the RBAC model.

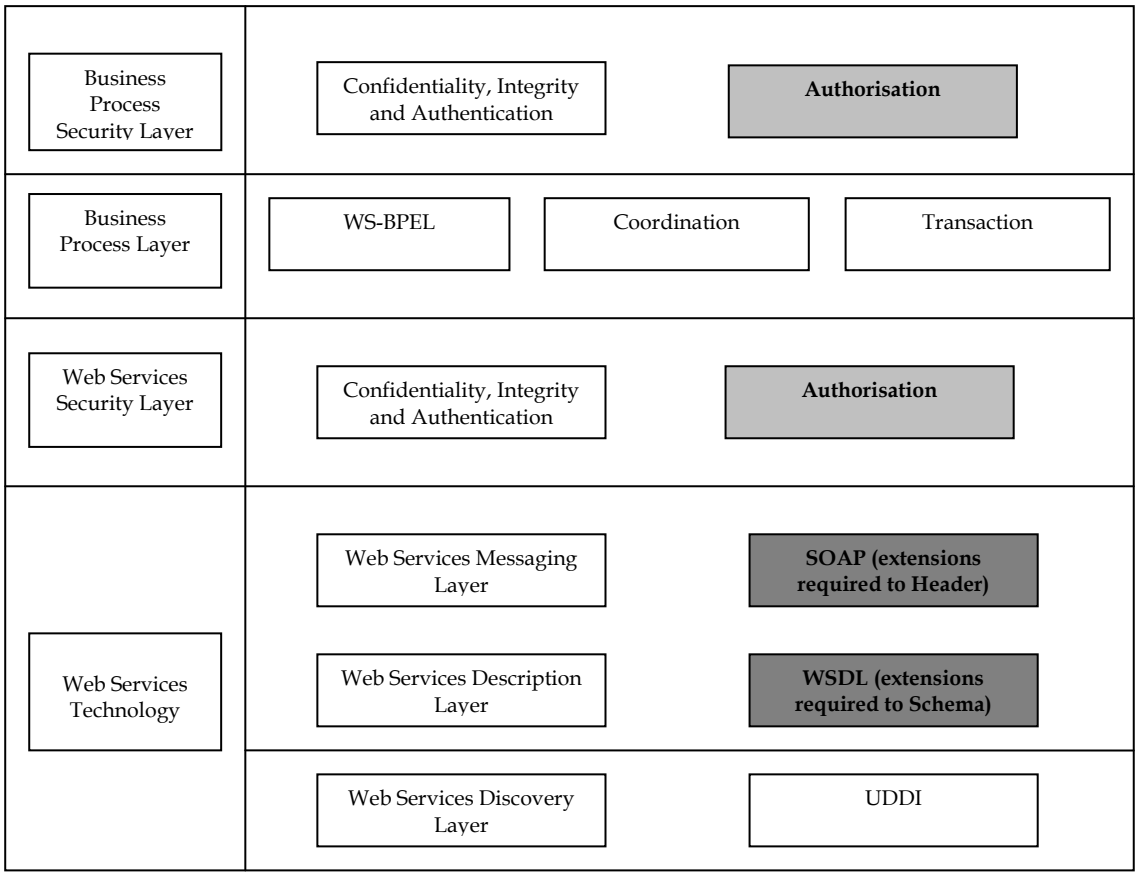

Figure 1. Layers in SOA (adopted from [14])

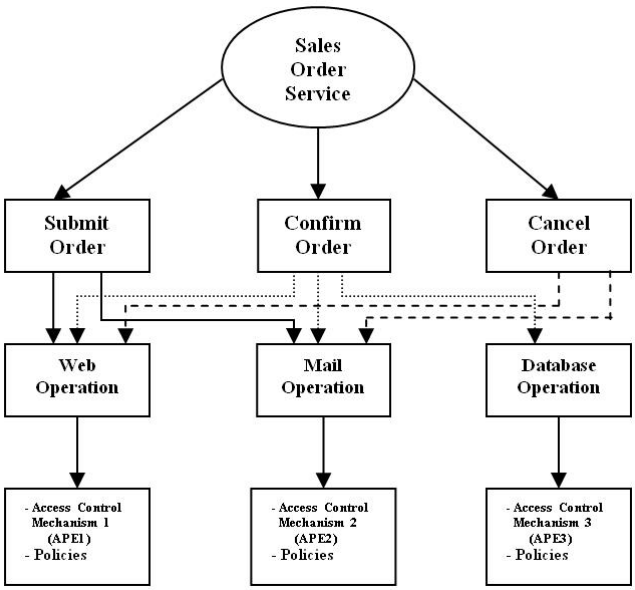

Figure 2. Sales Order service (adapted from [14])

An authorisation architecture for the business process layer of SOA must provide orchestration services to coordinate the authorisation decisions from an individual partner's authorisation policy evaluators. Each partner must be allowed to control its own authorisation policies and also not be required to disclose them to the entire workflow or to the workflow engine. Even in cases where the binding to actual end-points of partner services happens dynamically at runtime, the authorisation architecture must be able to orchestrate the partners' authorisation

policy evaluators and arrive at an authorisation decision. In the Command and Control (C2) use case shown in Figure 3, each of the partner services (agents) may potentially use their own access control mechanism. The partner agent (for example, Tactical Commander Agent (TCA)) may not wish to disclose its policies to the Regional Commander Agent (RCA). Similarly, other partners also may not wish to disclose their policies to be combined by the Strategic Commander Agent in order to authorise the client. In the course of the workflow, the commander (client) needs to get authorised seamlessly to partner services (agents) while delivering the Operational Orders as described in [18, 19].

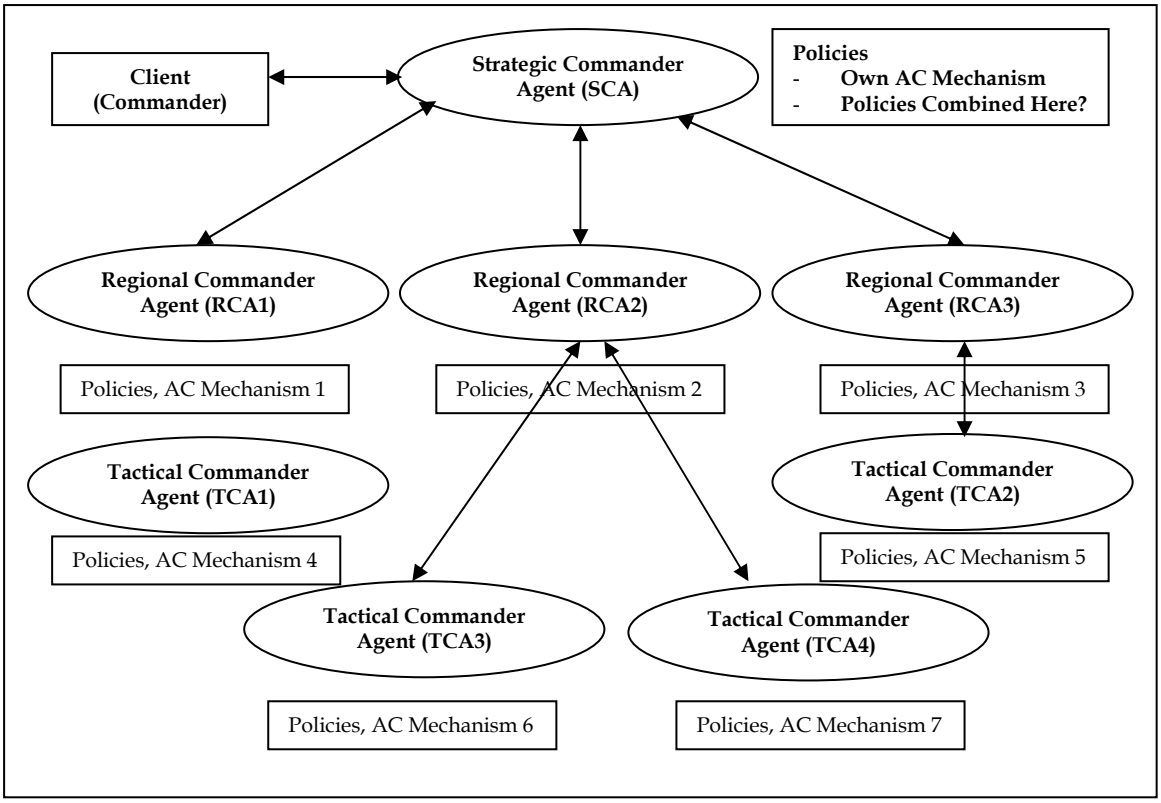

Figure 3. Command and Control Use Case (adapted from [18, 19])

Currently, there are a few authorisation schemes that are designed either for the Web services layer [10-12, 20, 21] or the business process layer [22-25] of SOA. After carrying out a survey [26] and analysis of the existing authorisation frameworks built for SOA and other distributed systems, we understand the design requirements for authorisation services for the Web service and business process layers of SOA. In this report, we propose a comprehensive authorisation framework for SOA by taking into account the design requirements we proposed in [27, 28]. We also compare related work to our authorisation framework in [26].

Our authorisation framework for SOA comprises the Web Services Authorisation Architecture (WSAA) [21, 29], built for the Web services layer of SOA and, the Business Process Authorisation Architecture (BPAA) [25], built for the business processes layer of SOA. BPAA leverages WSAA and extends its functionality. Section 3 describes the design of the WSAA and Section 4 describes the design of the BPAA. The WSAA and BPAA are indicated by the light-grey coloured boxes in Figure 1. Extensions to the Web services description and messaging layers are also proposed to support the unified SOA authorisation framework; these extensions are indicated by the dark-grey coloured boxes in Figure 1.

3. Design of the Web Services Authorisation Architecture (WSAA)

The WSAA comprises an administrative domain and a runtime domain (see Figure 4). We manage Web services in the administration domain by arranging them into collections and the collections themselves into a hierarchy. We provide administration support to manage a collection of Web services. We also provide support for the arrangement (adding, removing) of Web services within the collections and the movement of Web services within collections. Authorisation related components such as authorisation policy evaluators (evaluate authorisation policies), certificate and credential authorities (provide authentication certificates and authorisation credentials) and dynamic attribute services (provide attributes required at runtime for authorisation) can be managed in the administration domain. Also security administrators can assign a set of authorisation policy evaluators to authorise requests to Web services.

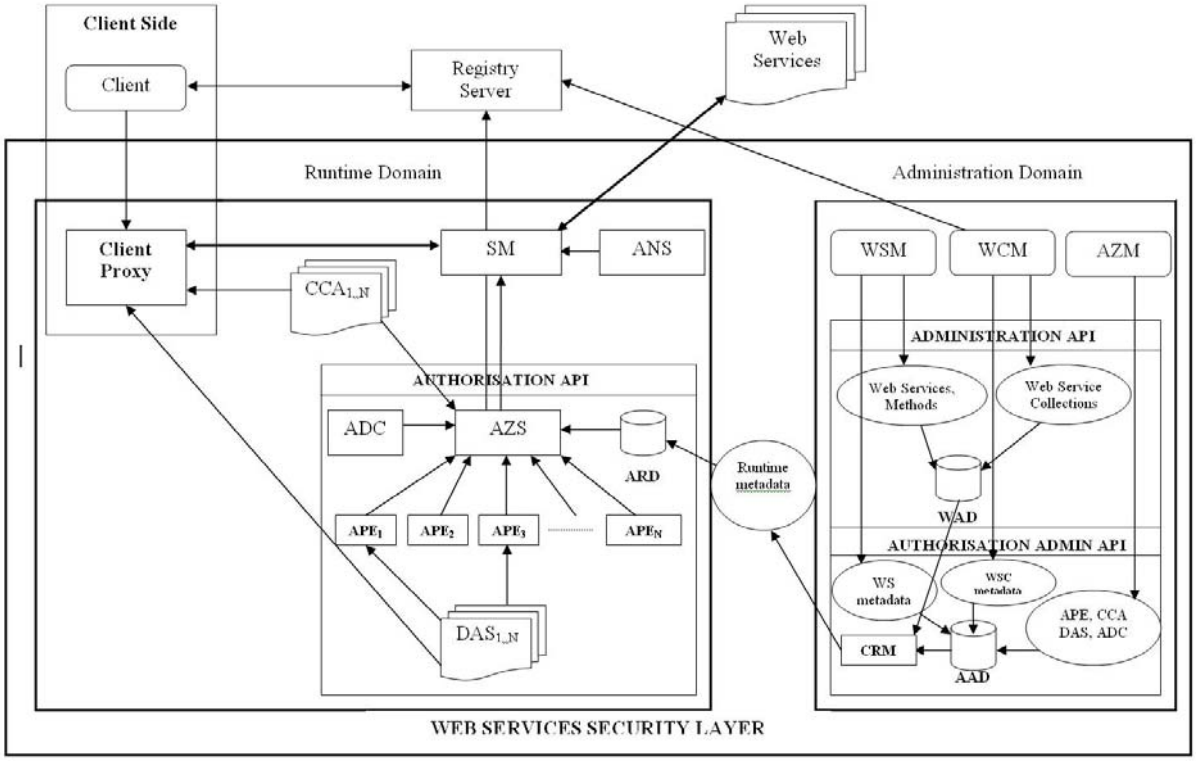

Figure 4. Web Services Authorisation Architecture (WSAA) (adopted from [14])

To make the authorisation process efficient, we have a runtime domain where the authorisation related information such as what credentials are required to invoke a particular Web service and how to collect those credentials, is compiled and stored. This information is automatically compiled from time to time when necessary using the information from the administration domain and it can be readily used by the components in the runtime domain.

The Registry Server located anywhere in the Internet (see Figure 4) is responsible for maintaining relations between services and their service providers. When a client requests the Registry Server for a specific service, the latter responds with a list of Web services that implement the requested service. For example, a UDDI [30] directory is a Registry Server.

3.1 System Components

We define the set of Authorisation Policy Evaluators, Certificate and Credential Authorities, Dynamic Attribute Services, and Authorisation Decision Composers as objects in our system. The Authorisation Manager (AZM) for the organisation is responsible to manage these components. S/he uses the Authorisation Administration API to manage them and the related data is stored in the Authorisation Administration Database (AAD). These objects are formally defined in Definitions 1- 4 below.

Certificate and Credential Authority (CCA): A **CCA** is responsible to provide authentication certificates and/or authorisation credentials required to authenticate and/or authorise a client**.** For example, a CCA may provide authentication certificates such as X.509 certificates [31] or authorisation credentials such as a Role Membership Certificate (RMC) [32] or a Privilege Attribute Certificate (PAC) [33].

Dynamic Attribute Service (DAS): A **DAS** provides system and/or network attributes such as bandwidth usage and time of the day. A dynamic attribute may also express the properties of a subject that are not administered by security administrators. For example, a Security Officer may only access a Defence employee's record if they are located within a Defence site. A DAS may provide the Security Officer's 'location status' attribute at the time of access control. Dynamic attributes' values change more frequently than traditional static authorisation credentials (also called privilege attributes) such as a Role Membership Certificate. Unlike authorisation credentials, dynamic attributes must be obtained at the time an access decision is required and their values may change within a session.

Authorisation Policy Evaluator (APE): An **APE** is responsible for making an authorisation decision on one or more abstract system operations. Every APE used by an organisation may use a different access control mechanism and a different policy language to specify authorisation policies. However, we define a standard interface for the set of input parameters an APE expects (such as subject identification, object information, and the authorisation credentials) and the output authorisation result it provides.

Authorisation Decision Composer (ADC): An **ADC** combines the authorisation decisions from APEs using an algorithm that resolves authorisation decision conflicts and combines them into a final decision.

Definition 1. Certificate and Credential Authority

We define Certificate and Credential Authority as a tuple cca = {i, l, CR, pa, ra(pa)} where i is a URN, l is a string over an alphabet Σ^* representing a network location such as a URL, CR is the set of authentication certificates and/or authorisation credentials cca provides, pa is an input parameter representing a subject, ra uses pa and gives out an output (result) that is the set of credentials for the subject.

Definition 2. Dynamic Attribute Service

We define Dynamic Attribute Service as a tuple das = {i, l, AT, pd, rd(pd)}, where i is a URN, l is a string over an alphabet Σ^* representing a network location such as a URL, AT is the set of attributes that das provides, pd is input parameter(s) representing attribute(s) name(s), rd uses pd and gives out an output (result) that is the value of the attribute(s).

Definition 3. Authorisation Policy Evaluator

We define Authorisation Policy Evaluator as a tuple ape = {i, l, pe, re(pe), OP, DAS, CCA}, where i is a URN, l is a string over an alphabet Σ^* representing a network location such as a URL, pe is the set of input parameters such as subject and object details, re is a function that uses pe and gives out an output (result) of authorisation decision. OP is the set of abstract system operations for which ape is responsible. DAS is the set of Dynamic Attribute Services responsible for providing dynamic runtime attributes to ape. ape uses these attributes to make authorisation decisions. CCA is the set of Certificate and Credential Authorities that provide the credentials required by ape.

Definition 4. Authorisation Decision Composer

We define Authorisation Decision Composer as a tuple adc = {i, l, a, pc, rc(pc)}, where i is a URN, l is a string over an alphabet Σ^* representing a network location such as a URL, a is the name of a pre-defined algorithm adc uses to combine the decisions from the individual authorisation policy evaluators. pc is an input parameter representing the decisions from the individual Authorisation Policy Evaluators involved. rc uses pc and the algorithm a to combine the decisions and gives out an output (result) that is the value of the final authorisation decision.

The runtime domain consists of the Client Proxy, Security Manager, Authentication Server and the Authorisation Server components.

Client Proxy (CP) is an automated component that collects the required authentication certificates and authorisation credentials from the respective authorities on behalf of the client before sending a Web service request and handles the session on behalf of the client with a Web service's Security Manager. It runs on the client side.

Security Manager (SM) is a component responsible for both authentication and authorisation of the client. The SM receives the necessary authentication certificates and authorisation credentials from the CP and sends them to the Authentication Server and the Authorisation Server for evaluation. It is responsible for managing all the interactions with a client's CP.

Authentication Server (ANS) receives the authentication certificates from the SM and uses some mechanism to authenticate the client. We treat ANS as a black box in our architecture as our focus is on authorisation of the client. We included this component in the Web services security layer for completeness. Note that the authentication of a client is a prerequisite to their authorisation.

Authorisation Server (AZS) decouples the authorisation logic from application logic. It is responsible for locating all the Authorisation Policy Evaluators involved, sending the authorisation credentials to them and receiving the authorisation decisions. Once all the decisions come back, it uses the responsible Authorisation Decision Composers (ADCs) to combine the authorisation decisions. Where required, AZS also collects the credentials on behalf of clients from the respective Certificate and Credential Authorities (CCAs).

3.1.1 Web Services Model

We consider a Web service model where *Web Service*, *Web Service Method* and *Web Service Collection* are viewed as objects. Web service collections are used to group together a set of possibly related Web service objects. Authorisation related information can be managed in a convenient way if a set of related Web service objects is grouped together in a hierarchy of

collections. Hierarchical containment is widely used in current Internet information systems such as Web servers, file systems, URLs and is well understood. We formally define (Definitions 5-7) the Web Service, Web Service Method and Web Service Collection objects based on the model discussed in [34].

Definition 5. Web Service

We define a Web Service as a tuple $ws = \{i, b, l, \Sigma, OP_{ws}, M, MD, wsm, sm\}$, where i is a non-empty string over an alphabet Σ^* representing a globally unique identifier such as a URN, b is a string over an alphabet Σ^* representing a network protocol binding such as SOAP over HTTP, l is a string over an alphabet Σ^* representing a network location such as a URL, Σ is a finite set of states representing the internal state of the object at a given time, OP_{ws} is the set of abstract operations (for e.g. Database operation such as create table, read a row in a table, or File operation such as read and write) performed by the methods of the ws object. M is the set of supported Web service methods, MD is the metadata providing additional description for ws. wsm is the identity of the *Web Service Manager* responsible for managing the ws object. sm is the location of the Security Manager responsible for securing ws object. Σ, M, OP_{ws} or MD can be the empty set $\emptyset$.

Definition 6. Web Service Method

We define a Web Service Method as a tuple $m = \{i, ws, OP_m, pm, rm(pm), MD\}$, where i is a URN, ws is the Web service object the method belongs to, OP_m is the set of abstract operations the method m performs. OP_m is a subset of the set OP_{ws} defined in the ws object. pm is the set of input parameters, string over an alphabet Σ^*, rm is a function $\Sigma^* \rightarrow \Sigma^*$ that maps pm onto a result string over an alphabet Σ^* representing the output (result) or return value(s) of a computation. pm and $rm(pm)$ may be the empty string ϵ. MD is a set of metadata providing additional description for method m. OP_m or MD can be the empty set $\emptyset$. A method m has to be a member of exactly one ws.

Definition 7. Web Service Collection

We define a Web Service Collection (WSC) as a tuple $wsc = \{i, WS, WSC_{CHILDREN}, p, MD, wcm, sm\}$, where i is a URN, WS is a finite set of (possibly related) Web service objects in wsc, $WSC_{CHILDREN}$ is a finite set of Web service collections that are children of wsc, p is the parent WSC (a WSC can have only one parent collection), MD is a finite set of metadata providing additional description and semantics for wsc. wcm is the identity of the *Web service Collection Manager* responsible for wsc. sm is the location of the Security Manager responsible for wsc. sm is null for all Web service collections in a hierarchy except for the root Web service collection, or the one without a parent p. In other words, if a wsc object has a parent p, it does not have a Security Manager. A root WSC's Security Manager is responsible for authentication and authorisation of requests to all the Web services (Web service objects) under its descendant collections. Figure 5 shows an example of a hierarchy of Web service collections.

3.1.2 Web Services Administration

In this section, we discuss the administration support provided by the WSAA to manage a collection of Web services.

A Web Service Manager (WSM) manages Web Services and Web Service Methods. A Web service Collection Manager (WCM) manages Web Service Collections using the *Administration API* (see Figure 4). These objects are stored in the Web service Administration Database (WAD).

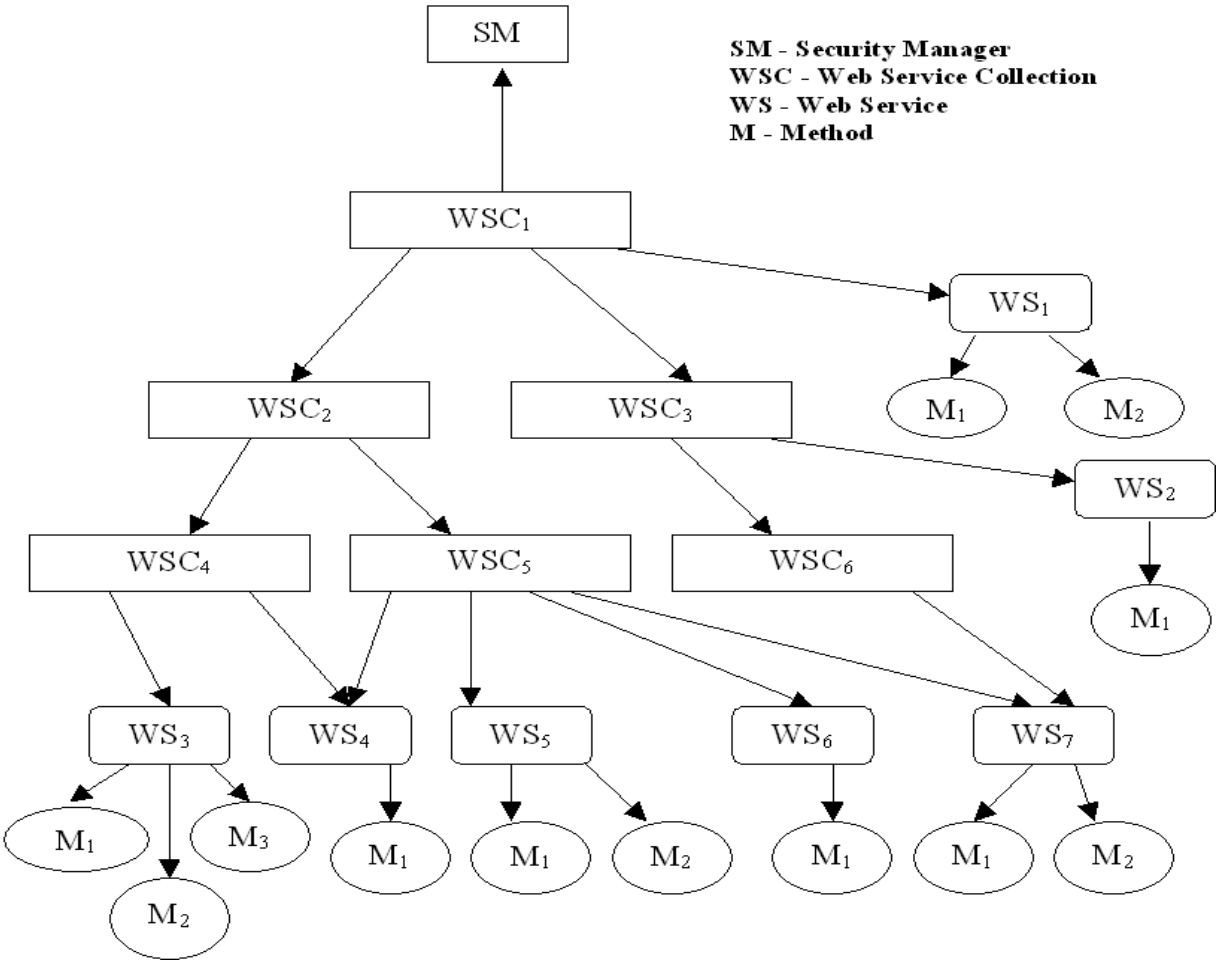

Figure 5. Web Service Collection Hierarchy (adopted from [14])

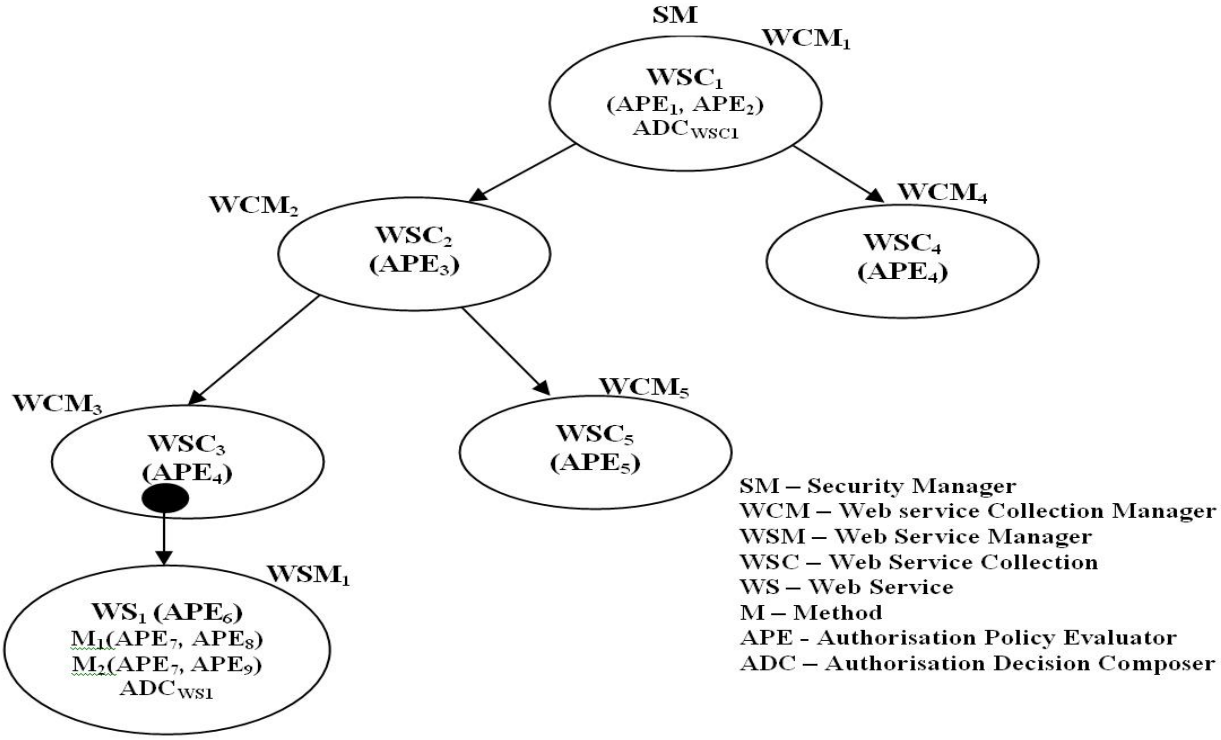

Figure 6. Web Services Collection Hierarchy (adopted from [14])

To effectively manage the collections, we arrange a set of related Web Service Collection (WSC) objects in a tree-shaped hierarchy as shown in Figure 6. Each WSC in the hierarchy has a responsible Web service Collection Manager (WCM). Now the following questions arise. Who can create the hierarchy structure? Who can add and delete Web Service (WS) objects to/from a WSC? Who has the privilege to move a WS object from one WSC to another? Before we answer these questions, we first define a root WCM.

In a WSC hierarchy tree, the root WSC's manager is called the *Root Web service Collection Manager* (RWCM). A RWCM is responsible for providing the Security Manager details in the WSDL statement of every Web service located under the collections it manages. RWCM provides these details using our extensions (discussed in Section 3.2) provided to the Web service description layer.

Let us now consider an organisation with a single hierarchy (such as the one shown in Figure 6) of Web service collections. In Figure 6, the root WSC is WSC_1 and the RWCM is WCM_1. We can consider a newly initiated system to simply consist of the root WSC, WSC_1 and a few WS objects under it managed by WCM_1. Now what can WCM_1 do with WS objects under WSC_1? S/he can add new WS objects from WAD into WSC_1. S/he can delete or move WS objects within the collections it is responsible for. Now there are two basic issues to consider.
1) Who decides the location of a WS object (and how is the location changed)?
2) Who decides the shape of the tree itself?

3.1.2.1 Web service object location

Consider the example shown in Figure 6. A WS object WS_1 resides in WSC_3 managed by WCM_3. Who has the authority to move a WS object from one WSC to another? For example, who has the authority to move WS_1 from WSC_3 to WSC_5? We can immediately rule out WCM_4 and WCM_5. WCM_5 has no authority over WS_1 at all (even if the point of the exercise is to give it the

authority). A WCM should not be able to arbitrarily assume authority over another WSC's objects. Can WCM_3 make the transfer? WCM_3's ability to do the transfer is a viable option. WCM_1 and WCM_2 may also be able to make the change. WS_1 resides in their descendant collection WSC_3. So it could be argued that they should be able to carry out the operation. In fact, it could be argued that only they could do it, as once a WCM is given responsibility for a WS object, they should not divest themselves of it. Hence WCM_3 should not be able to carry out the operation. Alternately, it may be desired to restrict the authority to make all such changes to only a single entity, in this case, the RWCM - WCM_1. So we have three possible options, each reflecting different organisational approaches:

a. The WCMs of the ancestor WSCs of the WSC in which the WS object being moved resides
b. Both the WCM of the immediate WSC in which the WS object resides and the WCMs of the ancestor WSCs of the WSC in which the WS object resides
c. The RWCM

It could be argued that a fourth option exists, that only the WCM of the immediate WSC (in this case WCM_3) can move the object. But as the meaning of the hierarchy is that a WS object resides in both its immediate WSC and in the ancestor WSC, this seems counter-intuitive. In the example, if WCM_3 is to have the authority, then it should not be placed as a subordinate to the other WCMs.

Option a presents a problem in the case of WS objects under a root WSC, as there are no ancestors in this case. This can be solved either by not allowing any WS objects to actually reside in the root WSC or by making an exception for the root WSC (effectively combining options a and c). In some sense the RWCM is a super user. Some organisations may not wish to have such a user, but for our purposes, this could be handled by having multiple WSC hierarchies in an organisation each managed by different RWCM.

The next question is, are there any limits to the destination of the WS object while making a move within the hierarchy? The most restrictive is to allow a WCM to move a WS object to a WSC under its control (the immediate WSC managed and other descendant WSCs). Less restrictive options are to allow a WS object to be passed to the parent WSC or any other ancestral WSCs. The target WSC to which a WS object is moved in the tree may be:

1. Any WSC in the tree
2. The WSC of the WCM making the move or any descendant WSC
3. As in 2 plus the immediate parent WSC
4. As in 2 plus any ancestor WSCs

Combining this with the example above, we would have the rule that a WCM can relocate any WS object that resides in their immediate WSC or any descendant WSCs to a target WSC that is either the WSC managed, any descendant WSC or any ancestor WSC. Note that this also gives us a rule placing a new WS object into a WSC. Let us again consider our example from Figure 6 of moving WS_1 object from WSC_3 to WSC_5. Under rule 3 above, only WCM_1 or WCM_2 could perform the operation. Only under option 2 on which WCM can make the move and option 1 on destination could WSM_3 move the WS_1 object. Even with option 2 on who can make the move, any other choice of options but option 1 for destination control prevents WCM_3 from making the move. Another possible option might be that WS objects could be transferred to sibling WSCs. This would then allow WCM_3 to move WS_1 object from WSC_3 to WSC_5. We find it likely that some organisations would not allow such transfers of responsibility without the involvement of more senior authority. However, it would allow WSC management structures to be created where WS objects could be transferred between the WSCs of a specified group of WCMs without allowing unrestricted transfers to any point in the hierarchy tree. This could be achieved with the combination of any of the options 2, 3 and 4 above.

3.1.2.2 *Shape of the tree*

Let us now briefly consider the creation and deletion of WSCs and the appointment of WCMs for the WSCs. Initially, a root WSC with a RWCM is created. RWCM appoints himself/herself to the root WSC. Organisations could also use a simple rule where manipulating the shape of the tree be restricted to the RWCM. A similar rule can apply to deleting WSCs from a tree. In this case, the RWCM appoints the WCMs for all the WSCs in the tree.

In a less restrictive scenario, any WCM in the tree should only be able to create and delete the descendant WSCs as well as appoint a WCM for the WSC s/he manages. Manipulating the WSCs for which the WCM has no responsibility is obviously undesirable.

3.1.3 Authorisation Administration and Policy Evaluation

A *Web Service Manager* (WSM) is responsible to manage the authorisation related information for the Web services s/he is responsible for. We consider a Web service method to be a high-level task that is exposed to clients. Each task (method) is made up of a number of system operations. These operations can be of different abstract types. For instance, each method of the Sales Order service shown in Figure 2 performs one or more of these three operations – Web operation, Database operation and Mail operation. Each of these operations has a responsible Authorisation Policy Evaluator (APE). Web operations are authorised by using the Java Authentication and Authorisation Service; Database operations are authorised by the Privilege Management Infrastructure (PMI); and the Mail operations are authorised by using a Role-based Access Control (RBAC) mechanism. It is reasonable to assume a WSM knows the set of organisation-defined tasks (such as Web operation in the case of Sales Order Service) a Web service under his/her control performs. Similarly a WSM knows the set of operations each of these tasks (methods) perform. The WSM associates these operations to the Web services and their methods. Then using the APE definitions from the Authorisation Administration Database (AAD), the administration domain automatically associates APEs to Web service methods. The association to APEs happens based on the operations the WSM associates with Web service methods. For instance, suppose that APE_1 provides authorisation for Web operations and APE_2 provides authorisation for mail operations, and the WSM chooses Web operation and mail operation as operations for the 'submit order' method of Sales Order Service. In this case, APE_1 and APE_2 are automatically associated with the 'submit order' method of Sales Order Service. This association is made in the *Web Service Method Authorisation* (WSMA) object. The object is stored in the AAD (see Figure 4). Therefore, we are able to separate authorisation responsibilities from Web Service Managers who are typically involved in developing Web services.

Similar to Web service methods, a Web service can also have one or more APEs responsible for Web service level authorisation. A WSM may associate one or more APEs to a Web service manually should s/he decide to do so. This association is made in the *Web Service Authorisation* (WSA) object. The object is stored in the AAD (see Figure 4). We give this provision to enable WSMs (or Web service developers) to associate coarse-grained authorisation with the Web services they manage. For instance, APE_3 can be associated to Sales Order Service. Web service level policies are first evaluated before its method level authorisation policies are evaluated. A Web service's APEs evaluate Web service level authorisation policies. These policies will typically be relatively coarse-grained and not be as fine-grained as method level authorisation policies. A WSM may choose to create a new Authorisation Decision Composer (ADC) for one or more Web services s/he manages or may decide to use one from the set of existing ADCs from the AAD if it serves the purpose.

Similar to Web services and their methods, a Web service collection can also have one or more APEs responsible for authorising access to the collection itself. A WCM may associate one or more APEs to a Web service collection manually should s/he decide to do so. This association is made in the *Web Service Collection Authorisation* (WSCA) object. The object is stored in the AAD. We give this provision to enable WCMs to associate coarse-grained authorisation with the Web service collections they manage. Collection level policies are first evaluated before Web service level authorisation policies are evaluated. A Web service collection's APEs evaluate collection level authorisation policies. These policies will typically be coarse-grained when compared to Web service and Web service method level policies. Every root Web service collection has an ADC associated with it responsible for combining the decisions from all the APEs involved. The coarse-grained authorisation policies for all relevant ancestor Web service collections (of an invoked Web service) are first evaluated, followed by the Web service level authorisation policies and finally the fine-grained Web service method level policies are evaluated. The coarse-grained policies are first evaluated before the finer-grained policies as it helps reduce the computing cost. If the client is not authorised by a coarse-grained policy, access can be denied straight away. For example in Figure 6, when a client invokes WS_1's method M_1, WSC_1's authorisation policies are first evaluated by APE_1 and APE_2, followed by WSC_2 (APE_3) and then WSC_3 (APE_4) policies. If APE_1, APE_2, APE_3 and APE_4 give out a positive decision, WS_1's authorisation policies are evaluated by APE_6. If APE_6 gives out a positive decision, then finally M_1's authorisation policies are evaluated by APE_7 and APE_8. WS_1's Authorisation Decision Composer, ADC_{WS1} combines the decisions from APE_6, APE_7 and APE_8 and if the final decision is positive, WSC_1's authorisation decision composer, ADC_{WSC1} combines the decisions from APE_1, APE_2, APE_3, APE_4 and ADC_{WS1}. If the final decision from ADC_{WSC1} is positive, the client will be authorised to invoke WS_1's method M_1.

WSMs manage WSA and WSMA using the Authorisation Administration API. Similarly WCMs manage WSCA objects using the Authorisation Administration API. WSA, WSMA and WSCA objects are stored in the AAD. We formally define WSA, WSMA and WSCA objects in Definitions 8–10.

Definition 8. Web Service Method Authorisation

We define Web Service Method Authorisation as a tuple $wsma = \{i, m, APE_m\}$, where i is a URN, m is the URN of the method for which the $wsma$ object is defined. APE_m is the set of URNs of the Authorisation Policy Evaluators responsible for authorising requests from a client to the method m.

Definition 9. Web Service Authorisation

We define Web Service Authorisation as a tuple $wsa = \{i, ws, APE_{ws}, adc_{ws}\}$, where i is a URN, ws is the URN of the Web Service for which wsa is defined. APE_{ws} is the set of URNs of the Authorisation Policy Evaluators responsible for authorising requests from a client to ws. adc_{ws} is the URN of the Authorisation Decision Composer for ws. It is responsible for combining the decisions from Authorisation Policy Evaluators in the set APE_{ws}.

Definition 10. Web Service Collection Authorisation

We define Web Service Collection Authorisation as a tuple $wsca = \{i, wsc, APE_{wsc}, adc_{root}\}$, where i is a URN, wsc is the URN of the Web Service Collection for which the $wsca$ object is defined. APE_{wsc} is the set of URNs of the Authorisation Policy Evaluators responsible for wsc. adc_{root} is the URN of the Authorisation Decision Composer for wsc. If wsc is not a root Web Service Collection, then adc_{root} is null. In other words, adc_{root} exists only for a root wsc.

3.1.4 Runtime Authorisation Data

APEs and ADCs have now been assigned to Web services and Web service collections. The next question is, at runtime, how does a client know (where necessary) how to obtain the required authorisation credentials and dynamic runtime attributes before invoking a Web service? What are the responsible APEs (and the credentials and attributes they require), CCAs (the credentials they provide) and the DASs (the attributes they provide)? How does the AZS know what the set of responsible ADCs (adc_{ws} and adc_{root}) for a particular client request is?

To answer these questions, we have an Authorisation Runtime Database (ARD) in the runtime domain. ARD consists of the runtime authorisation related information required by the clients (Client Proxies) and the Authorisation Server. The *Credential Manager* (CRM) is an automated component that creates and stores the authorisation runtime information in ARD using the information from the WAD and AAD databases. The runtime authorisation information consists of three tuples defined in Definitions 11–13. The CRM is invoked from time to time, when a Web service object is added to or deleted from a collection, moved within a hierarchy of collections or when the shape of the tree itself changes, to update these tuples in the ARD.

Definition 11. Method-Credential-CCA tuple

We define the Method-Credential-CCA tuple as *mcc* = {*i, m, CR, cca, ape*}, where *i* is a URN, *m* is a Web service method to which the tuple is defined, *CR* is the set of authorisation credentials to be obtained from the Certificate and Credential Authority *cca* to be authorised to invoke *m*. *ape* is the Authorisation Policy Evaluator that requires these credentials. This means each *m* object can have one or more of these (tuple) entries in the ARD, as an APE may need credentials from more than one CCA, and more than one APE may control access to the method.

Definition 12. Method-Attribute-DAS tuple

We define the Method-Attribute-DAS tuple as *matd* = {*i, m, AT, das, ape*}, where *i* is a URN, *m* is a Web service method to which the tuple is defined, *AT* is the set of attributes to be obtained from the Dynamic Attribute Service *das*. Each *m* object can have one or more of these (tuple) entries in ARD. *ape* is the Authorisation Policy Evaluator that requires these attributes.

Definition 13. WS-ADC tuple

We define the WS-ADC tuple as *wsd* = {*i*, *ws*, adc_{ws}, adc_{root}}, where *i* is a URN, *ws* is a Web service, adc_{ws} is the Authorisation Decision Composer for *ws*. adc_{root} is the Authorisation Decision Composer for the root Web service collection in which *ws* resides.

3.1.4.1 CRM algorithm

CRM creates these tuple entries (from Definitions 11–13) in the Authorisation Runtime Database (ARD) using the following algorithm:

1. For every Web service method defined in the WAD, it generates a list, APE_{List} of responsible authorisation policy evaluators using its *wsma* tuple and related *wsa* and *wsca* tuples from the AAD,
2. For each Authorisation Policy Evaluator, ape in APE_{List},
    a. For each cca responsible for the ape, an *mcc* tuple is created in the ARD.
    b. For each das the ape uses, an *matd* tuple is created in the ARD.
3. For every Web service object, CRM creates a *wsd* tuple entry in the ARD using the *wsa*, *wsc* and *wsca* tuples of the root Web service collection in which the Web service object resides.

When a Web service object is placed and/or moved within a Web service collection in a tree, the set of Authorisation Policy Evaluators responsible for authorising a client's requests changes. Similarly, the set of Certificate and Credential Authorities and Dynamic Attribute Services responsible also changes. For example, in Figure 6, when WS_1 moves from WSC_3 to WSC_5, the set of responsible Authorisation Policy Evaluators for WS_1's method M_2 changes from {APE_1, APE_2, APE_3, **APE_4**, APE_6, APE_7, APE_9}f to {APE_1, APE_2, APE_3, **APE_5**, APE_6, APE_7, APE_9}. Once the move is made, CRM is automatically invoked and it updates the ARD with the necessary *mcc* and *matd* tuple entries for each method of WS_1. In this example, the *wsd* tuple remains the same before and after the update. The responsible Authorisation Decision Composers before and after the move will still be ADC_{WSC1} and ADC_{WS1}.

3.1.5 Authorisation Algorithms

The WSAA supports three authorisation algorithms. The first, *push-model* algorithm supports authorisations where a client's Client Proxy, using the information in WS-AuthorisationPolicy, collects and sends the required authorisation credentials (from Certificate and Credential Authorities) and attributes (from Dynamic Attribute Services) to a Web service's Security Manager. The second, *pull-model* algorithm supports authorisations where the Authorisation Server (AZS) itself collects the required authorisation credentials from Certificate and Credential Authorities and the Authorisation Policy Evaluators (APEs) themselves collect the required attributes from Dynamic Attribute Services. The AZS in this case uses the runtime objects (tuples) information from the Authorisation Runtime Database to be able to do so. The third, *combination-model* supports both the push and pull models of collecting the required authorisation credentials and attributes.

When the combination-model algorithm is deployed by an organisation, the organisation's Authorisation Manager (AZM) may arbitrarily decide whether the authorisation credentials required from a Certificate and Credential Authority and dynamic attributes required from a Dynamic Attribute Service for each Web service method, Web service as well as Web service collection level APEs, are fetched by a Client Proxy (push-model) or by the authorisation components themselves (pull-model). The AZM may decide to give the entire responsibility of fetching the required credentials and attributes to the Client Proxy or to authorisation components or share responsibility of fetching credentials and attributes amongst the Client Proxy and the authorisation components. This information is reflected in a Web service's WS-AuthorisationPolicy, defined in Section 3.2.1.

3.1.6 Sequence of steps involved in Authorisation

We show the sequence of steps involved in authorising clients to Web services, using a system sequence diagram (see Figure 7). Various objects that participate in the interaction are placed in the sequence diagram across the X-axis. The object that initiates the action is placed on the left of the diagram, and increasingly more subordinate objects to the right. The messages that these objects send and receive are shown along the Y-axis, in order of increasing time from top to bottom. The Combination model authorisation algorithm shown in Figure 7. Steps involved in the Push and the Pull models of authorisation are not shown in Figure 7, as they are a subset of the Combination model authorisation algorithm. In the Push Model, we do not have steps 13.1, 14, 15, 17, and 18 and in the Pull Model we do not have steps 6 and 7. However in the Pull Model, we still have steps 4 and 5 to fetch the *authentication* certificates, where required.

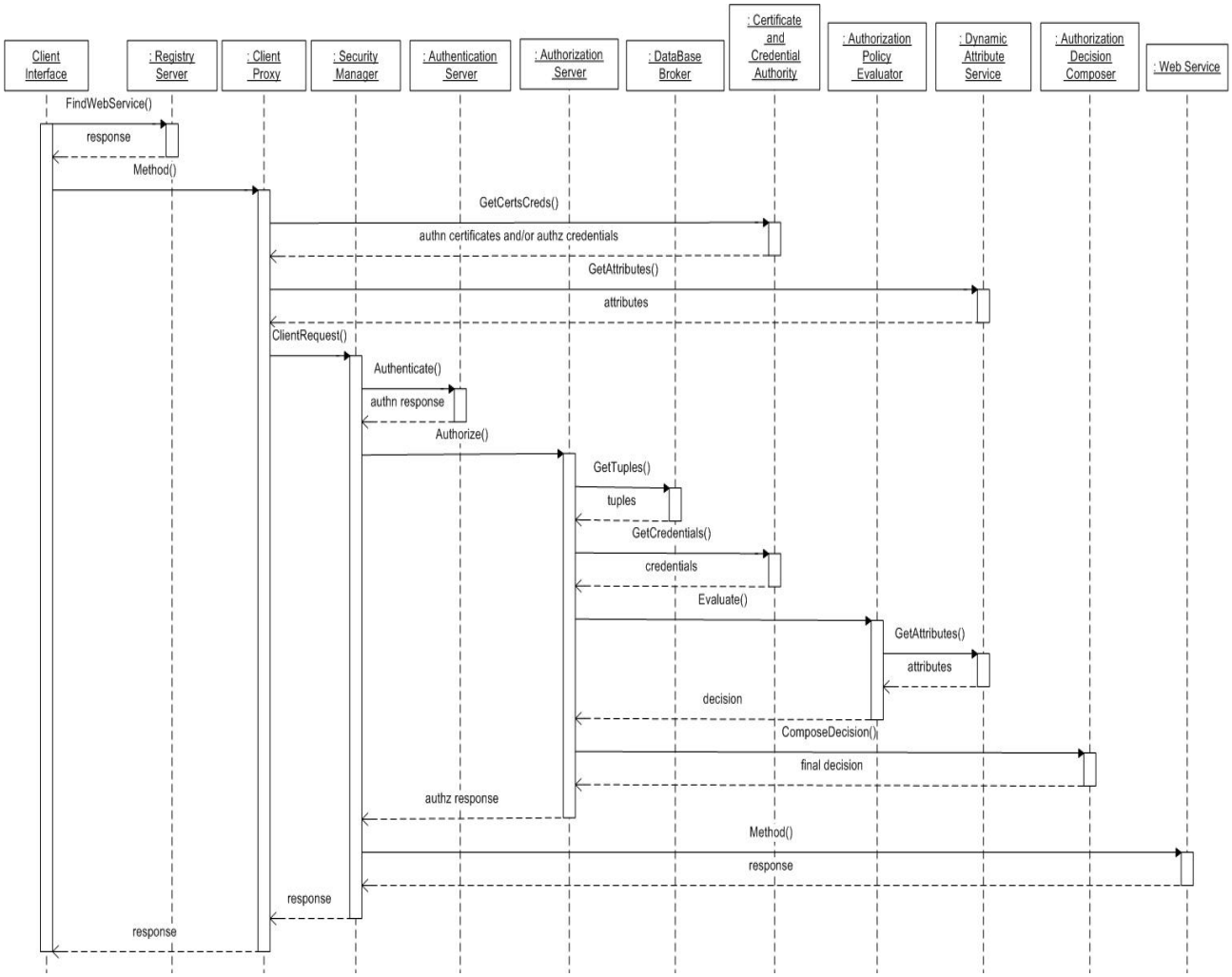

Figure 7. Sequence of steps involved in Combination Model Authorisation (adopted from [14])

(1) A client using an interface queries a Registry Server such as a UDDI directory for a Web service.
(2) The Registry Server looks up this Web service and returns a list of appropriate Web service(s).
(3) The client chooses a Web service WS_1 and invokes a method on WS_1.
(4) The Client Proxy intercepts the Web service request and retrieves WS_1's WSDL statement. It locates WS_1's WS-SecurityPolicy (for authentication) and WS-AuthorisationPolicy (for authorisation) statements. Using the information in WS-SecurityPolicy and WS-AuthorisationPolicy, the Client Proxy requests the required authentication certificates and/or authorisation credentials from a Certificate and Credential Authority.
(5) The Certificate and Credential Authority sends the required certificates or credentials to the Client Proxy. Steps 4 and 5 are repeated until all the required authentication certificates and/or authorisation credentials are collected.
(6) Using the information in WS-AuthorisationPolicy, the Client Proxy requests the required runtime attributes from a Dynamic Attribute Service.
(7) The Dynamic Attribute Service sends the required attributes to the Client Proxy. Steps 6 and 7 are repeated until all the required attributes are collected.
(8) The Client Proxy sends the Web service (method) request along with the collected authentication certificates, authorisation credentials and runtime attributes to WS_1's Security Manager. It finds the Security Manager location using WS_1's WSDL statement.
(9) The Security Manager sends the authentication certificates to the Authentication Server.

(10) The Authentication Server gets back with a decision. If authentication fails, then the Security Manager sends an "authentication fail" message to the Client Proxy and the access is denied to the client (this step is not shown in Figure 7). If authentication is successful, the algorithm continues as follows:

(11) The Security Manager sends the authorisation credentials and attributes received from the Client Proxy to the Authorisation Server using the Authorisation API.

(12) The Authorisation Server sends a request to the Database Broker component for the sets of tuples (runtime objects) for the requested Web service, WS_1. The Database Broker fetches these details from the Authorisation Runtime Database.

(13) The Authorisation Server receives the sets of requested tuples. Using them, the Authorisation Server creates a list of responsible Authorisation Policy Evaluators, APE_{List}, for the method requested.

(13.1) For each Authorisation Policy Evaluator in APE_{List}, the Authorisation Server creates a list of responsible Certificate and Credential Authorities, CCA_{List}.

(14) From each Certificate and Credential Authority in CCA_{List}, the Authorisation Server requests the set CR of necessary credentials on behalf of the client.

(15) The Certificate and Credential Authority sends the set CR of credentials to the Authorisation Server. Steps 14 and 15 are repeated until all the credentials are collected. These steps are only executed if the Client Proxy did not already send the credentials required by the Authorisation Policy Evaluator.

(16) The Authorisation Server, using the information in the tuples, sends the appropriate credentials to an Authorisation Policy Evaluator in the APE_{List}, along with the subject (client authentication details) and target (resource) details.

(17) The Authorisation Policy Evaluator, if necessary, requests a Dynamic Attribute Service to collect the set AT of runtime attributes required by its authorisation policies.

(18) The Dynamic Attribute Service sends the set AT of attributes to the Authorisation Policy Evaluator. Steps 17 and 18 are repeated to collect the required runtime attributes from all the Dynamic Attribute Services involved. They are executed if the Client Proxy did not already collect and send the required attributes.

(19) The Authorisation Policy Evaluator evaluates its authorisation policies using the credentials and/or attributes and then sends its authorisation decision to the Authorisation Server. Steps 13.1 to 19 are repeated for all the Authorisation Policy Evaluators in APE_{List}.

(20) The Authorisation Server locates the root Web service collection's Authorisation Decision Composer, adc_{root} using the necessary tuple (received from the Database Broker) and sends the set of authorisation decisions from all the Authorisation Policy Evaluators involved to it.

(21) The Authorisation Decision Composer, adc_{root}, combines the decisions using a pre-defined algorithm and sends the final authorisation decision to the Authorisation Server. As mentioned earlier in Section 4.4, adc_{root} first delegates the task of combining the Web service and Web service method level authorisation decisions to WS_1's Authorisation Decision Composer, adc_{WS1} and then combines that decision with Web service collection level authorisation decisions.

(22) The Authorisation Server gets back to the Security Manager with the final authorisation decision. If authorisation fails, then the Security Manager sends an "authorisation fail" message to the Client Proxy and the access is denied to the client (this step is not shown in Figure 7). If authorisation is successful, the algorithm continues as follows:

(23) The Security Manager acts as a broker for the client's request and sends the request to the appropriate Web service (WS_1).

(24) WS_1 gets back with a result to the Security Manager.

(25) The Security Manager sends the result back to the Client Proxy.

(26) The client (interface) receives the final result from the Web service via the Client Proxy.

3.2 Extensions to the Web Service Description and Messaging Layers

We require extensions to the Web Service Description and Messaging layers of SOA to support WSAA. We provide extensions to the SOAP header (messaging layer) to carry authorisation related credentials and attributes. We extend WSDL (description layer) to include a Web service's Authorisation Policy as well as the location of its Security Manager.

3.2.1 WS-AuthorisationPolicy statement

WS-SecurityPolicy [35] is a statement consisting of a group of security policy "assertions", that represent a Web Service's security preference, requirement, capability or other property. Similarly, we define *WS-AuthorisationPolicy* as a statement that contains a list of authorisation assertions. The assertions include what credentials (and from which Certificate and Credential Authority) and attributes (and from which Dynamic Attribute Service) a client's Client Proxy has to collect before invoking a Web Service. The WS-PolicyAttachment standard [36] can be used to link the WS-AuthorisationPolicy to a Web Service's WSDL statement. Figure 8 shows the XML schema skeleton for WS-AuthorisationPolicy.

```
<WS-AuthorizationPolicy>                          // for Web Service, WS1
    <URN>1                                        // WS1's URN
    <URL>1                                        // WS1's URL
    <WSDL-URL>1                                   // WSDL location for WS1
    <WS-Method >1..*                              // list of WS1 methods
        <WS-Method-URN/>1                         // method identification
        <Credentials>                             // list of credentials required
            <CCA>1..*                             // Certificate and credential authority
                <CCA-URL/>1                       // location of CCA
                <Credential/>1..*                 //credentials provided by CCA
            </CCA>
        </Credentials>
        <Attributes>                              // list of attributes required
            <DAS>1..*                             //Dynamic attribute service
                <DAS-URL/>1                       // location of DAS
                <Attribute/>1..*                  //attributes provided by DAS
            </DAS>
        </Attributes >
    </ WS-Method >
</ WS-AuthorizationPolicy >
```

Figure 8. WS-AuthorisationPolicy XML Schema Skeleton (adopted from [14])

```
<securityManager>1..1
        <location>URL</location>
</securityManager>
XML schema for Security Manager

<?xml version="1.0"?>
<definitions name="StockQuote"
            ....
            <securityManager>
                        <location>http://example.com</location>
            </securityManager>
            <types>
                        ...
            </types>
            <message name="GetLastTradePriceInput">
                        ...
            </message>
            <portType name="StockQuotePortType">
                        ...
            </portType>
            <binding name="StockQuoteSoapBinding"
                                    type="tns:StockQuotePortType">
                  ...
            </binding>
            <service name="StockQuoteService">
                  ...
            </service>
</definitions>
An example WSDL document with the extension.
```

Figure 9. Extended WSDL schema skeleton example (adopted from [14])

3.2.2 Security Manager Location

When a client wants to invoke a Web service WS_1, its Client Proxy requires its Security Manager's location. Therefore, we need to give this information in WS_1's WSDL statement. We introduce a new element called *SecurityManager* to the WSDL document. The XML schema skeleton for Security Manager element and an example WSDL statement are shown in Figure 9.

3.2.3 SOAP Header Extension

WS-Security [37] enhancements for confidentiality, integrity and authentication of messages have extended the SOAP header (SOAP-SEC element) to carry related information. Similarly we propose an extension to the SOAP header to carry authorisation credentials and attributes to carry authorisation related information. When a client wants to invoke a Web service object, its Client Proxy creates an authorisation header object and adds it to SOAP header before making a SOAP request. We show the XML schema skeleton for the extended SOAP header in Figure 10. The SOAP-AUTHZ header (shown in bold) consists of the list of credentials and attributes the Client Proxy collects on behalf of the client.

```
<SOAP-ENV>
        <SOAP-ENV:Header>
                <SOAP-SEC/>            // for confidentiality, integrity and authentication
                <SOAP-AUTHZ>           // for authorization
                        <Credentials>                       // list of collected credentials
                                <CCA>1..*                   // Certificate and credential authority
                                        <CCA-URN/>1         // CCA identification
                                        <Credential/>1..*   // credentials provided by the CCA
                                </CCA>
                        </Credentials>
                        <Attributes>                        // list of collected attributes
                                <DAS>1..*                   // Dynamic attribute service
                                        <DAS-URN/>1         // DAS identification
                                        <Attribute/>1..*    // attribute values provided by the DAS
                                </DAS>
                        </Attributes >
                </SOAP-AUTHZ>
        </SOAP-ENV:Header>

        <SOAP-BODY>
                ...
        </SOAP-BODY>
</SOAP-ENV>
```

Figure 10. XML Schema skeleton for SOAP Authorisation Header (adopted from [14])

3.3 Benefits of the WSAA

Some of the key advantages of the proposed WSAA architecture are as follows:

(a) Support for various access control models: The WSAA supports multiple access control models including MAC, DAC, and RBAC models. The access policy requirements for each model can be specified using its own policy language. The policies used for authorisation can be fine-grained or coarse-grained depending on the Web service requirements. Access control mechanisms can either use the push model or pull model or even a combination of both for collecting client credentials.

(b) Support for legacy applications and new Web service based applications: Existing legacy application systems can still function and use their current access control mechanisms when they are exposed as Web services to enable an interoperable heterogeneous environment. Once again, various access policy languages can be used to specify the access control rules for users. They could adopt a push or a pull model for collecting credentials. At the same time, the WSAA supports new Web service based applications built to leverage the benefits offered by SOA. New access control mechanisms can be implemented and used by both legacy and new Web service applications. A new access control mechanism can itself be implemented as a Web service. All WSAA requires is an end-point URL and interface for the mechanism's Authorisation Policy Evaluator.

(c) Decentralised and distributed architecture: A Web service can have one or more responsible Authorisation Policy Evaluators involved (each with its own end-point defined) in making the authorisation decision. The Authorisation Policy Evaluators themselves can be Web services specialising in authorisation. This feature allows the WSAA to be decentralized and distributed. Distributed authorisation architecture such as ours provides many advantages such as fault

tolerance and better scalability and outweighs its disadvantages such as more complexity and communication overhead.

(d) Flexibility in management and administration: Using the hierarchy approach of managing Web services and collections of Web services, authorisation policies can be specified at each level making it convenient for Web service collection managers (WCM) and Web service managers (WSM) to manage their objects as well as their authorisation related information.

(f) Ease of integration into platforms: Each of the components involved both in the administration and runtime domains is fairly generic and can be implemented in any middleware including the .NET platform as well as Java based platforms. The administration and runtime domain related APIs can be implemented in any of the available middleware. We have implemented the WSAA within the .NET framework and demonstrated the architecture using a healthcare application [29].

(g) Enhanced Security: In our architecture, every client request passes through the Web service's Security Manager and then gets authenticated and authorised. The Security Manager can be placed in a firewall zone, which enhances security of collections of Web service objects placed behind an organisation's firewall. This enables organisations to protect their Web service based applications from outside traffic. A firewall could be configured to accept and send only SOAP request messages with appropriate header and body to the responsible Security Manager to get authenticated and authorised.

4. Design of the Business Process Authorisation Architecture (BPAA)

We now discuss the design of our Business Process Authorisation Architecture (BPAA) suitable for the business process layer of SOA (see Figure 1). Before we delve into the design of the architecture, we clearly distinguish between *static* and *dynamic* business processes[2]. A static business process is a pre-composed business process, where all partner service interfaces and their binding information is known at design time itself. A dynamic business process is more complex, where only the partner interfaces are defined at design time, but not the actual bindings to real instances of partner services (Web services and/or business processes). The binding is made at runtime to real instances of services by letting the client interact with the business process. For instance, a travel agent may statically bind at design time to always book flight tickets with Qantas airlines, book cars with Hertz car rental, and finally a hotel room with Hilton. But in real-world situations, customers want more flexibility, and therefore, travel agents may opt to expose their services as dynamic business processes, where the customer at runtime chooses an appropriate partner service (such as airline, car rental or hotel) depending on their own requirements.

We make an important assumption in this report. A dynamic business process may not only invoke partner Web services but also partner services that are themselves business processes. We assume that such business processes are themselves static business processes. We make this assumption to keep the discussion relatively simple. However, the architecture has been extended to authorise clients to dynamic business processes that themselves invoke other partner dynamic business processes (in Section 4.4).

[2] In this report, when we mention "business process", we mean a business process defined using a WS-BPEL statement.

4.1 Business Processes Authorisation Architecture (BPAA)

Let us now first briefly describe an overview of the proposed architecture (see Figure 11). Similar to WSAA, BPAA comprises an administrative domain and a runtime domain. We manage business processes in the administration domain. Authorisation related components such as authorisation policy evaluators, certificate and credential authorities and dynamic attribute services can be managed in the administration domain. Also security administrators can assign a set of authorisation policy evaluators to authorise requests to business processes.

Once again, similar to WSAA, we have a runtime domain, where the authorisation related information such as what credentials are required to invoke a particular business process and how to collect those credentials, is compiled and stored. This makes the authorisation process efficient. This information is automatically compiled from time to time when necessary using the information from the administration domain and it can be readily used by components in the runtime domain. A client makes use of a registry server such as a UDDI directory to find business process definitions (WS-BPEL [15] statements).

4.2 Design of the Architecture

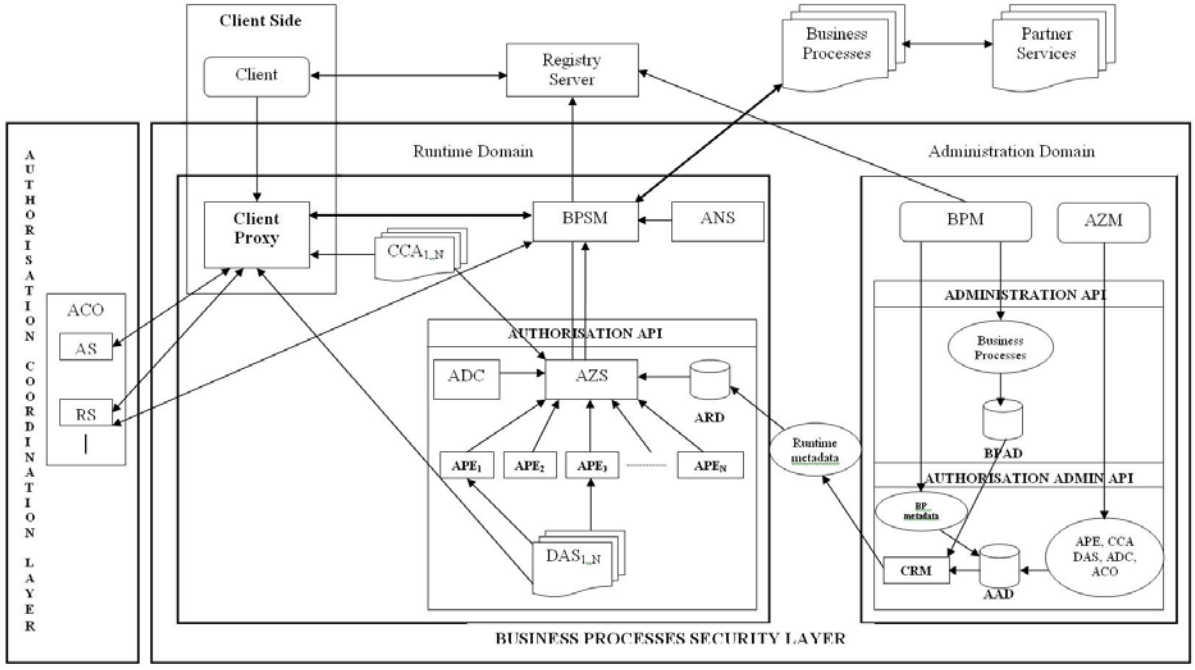

Figure 11. Business Processes Authorisation Architecture (BPAA) (adopted from [14])

4.2.1 System Components

We make use of the runtime components – Authorisation Policy Evaluators, Certificate and Credential Authorities, Dynamic Attribute Services and Authorisation Decision Composers in BPAA as well. We do not change their definitions in BPAA. For further details on these components, please see definitions 1 to 4 in Section 3.1. Once again, the Authorisation Manager (AZM) for the organisation is responsible for managing these components.

The runtime domain consists of the Client Proxy, Business Process Security Manager, Authentication Server, Authorisation Server, and the Authorisation Coordinator components.

Client Proxy (CP) collects the required authentication certificates and authorisation credentials from the respective authorities on behalf of the client before sending a request to a business process and handles the session on behalf of the client with a Business Process Security Manager.

Business Process Security Manager (BPSM) is responsible for both authentication and authorisation of the client to a business process. A client's Client Proxy sends the necessary authentication certificates and authorisation credentials to the BPSM. It is responsible for managing all the interactions with a client's Client Proxy.

Authentication Server (ANS) receives the authentication certificates from BPSM and uses some mechanism to authenticate the client. We treat ANS as a black box in our architecture as our focus in this report is on authorisation of the client. We included this component in the business processes security layer for completeness.

Authorisation Server (AZS) decouples the authorisation logic from the application logic. It is responsible for locating the business process' Authorisation Policy Evaluators (APEs), sending the credentials to them and receiving the authorisation decisions. Once all the decisions come back, it uses the business process' ADC to combine the authorisation decisions. If required, AZS also collects the required authorisation credentials on behalf of clients from the respective certificate and credential authorities.

Authorisation Coordinator (ACO) is used to coordinate authorisation between a client (by involving the Client Proxy) and *dynamic* business processes and their partner services (Web services and/or business processes). It comprises an **Activation Service** (AS) and a **Registration Service** (RS) that expose standard interfaces to the participants (Client Proxy and Business Process Security Manager) in the authorisation coordination protocol (based on the WS-Coordination [38] standard). We discuss the design of the authorisation coordination framework in Section 4.3.

4.2.2 Business Process Definition and Administration

A *Business Process Manager* (BPM) manages a set of business processes s/he is responsible for in an organisation. S/he uses the Administration API to manage the business processes. The business process definitions are stored in the *Business Process Administration Database* (BPAD). We define a business process in Definition 14.

Definition 14. Business Process

We define a Business Process as a tuple $bp = \{i, l, \Sigma, WS, BP, B, pa, MD, bpm, bpsm, aco\}$, where i is a non-empty string over an alphabet Σ^* representing a globally unique identifier such as a URN, l is a string over an alphabet Σ^* representing a network location such as a URL, Σ is a finite set of states representing the internal state of the business process at a given time, WS is the set of URNs of bp's partner Web services or activities, BP is the set of URNs of bp's partner business processes or activities, B is the network protocol binding such as SOAP over HTTP for the business process, pa represents the business process flow algorithm represented in a WS-BPEL statement, MD is the metadata providing additional description for bp, bpm is the identity (ID) of the Business Process Manager (BPM) responsible for managing bp. $bpsm$ is the location of the Business Process Security Manager component responsible for authentication and authorisation of clients to the business process. aco is the location of the Authorisation Coordinator responsible for coordinating authorisation of a client to bp's partner services. aco is defined only for dynamic business processes and is null for static business processes. Σ, B or MD can be the empty set Ø. If

B is an empty set, Ø, then the business process defined can either be an *abstract* business process or a dynamic business process. An abstract business process is not executable and it only defines the standard interfaces between a business process and its partner services and the messages passed between them. In a dynamic business process, individual bindings to partners are made at runtime using the client's preferences. If *B* is not an empty set at business process design time, then it is a pre-composed or static business process.

4.2.3 Authorisation Administration and Policy Evaluation

A Business Process Manager (BPM) manages the authorisation related information for the business processes s/he is responsible for. This information is stored in the *Business Process Authorisation* tuple. We define the tuple in Definition 15.

Definition 15. Business Process Authorisation

We define Business Process Authorisation as a tuple $bpa = \{i, bp, APE_{bp}, adc_{bp}\}$, where *i* is a URN, *bp* is the business process for which *bpa* is defined. APE_{bp} is the URNs of the set of Authorisation Policy Evaluators responsible for authorising requests from a client to *bp*. adc_{bp} is the URN of an Authorisation Decision Composer. It is responsible for combining at runtime, the authorisation decisions given out by the set of APEs in APE_{bp}.

4.2.4 Runtime Authorisation Data

Once again, just as in our WSAA, we have a Credential Manager (CRM) component in BPAA. It is responsible for compiling and storing the authorisation information required by components in the runtime domain. This runtime authorisation information is stored in the *Authorisation Runtime Database* (ARD). The runtime authorisation information consists of two tuples defined in Definitions 16 and 17. The CRM is invoked from time to time, when a business process object is created, modified or deleted from the BPAD.

Definition 16. BusinessProcess-Credential-CCA tuple

We define the BusinessProcess-Credential-CCA tuple as *pcc* = {*i, bp, CR, cca, ape*}, where *i* is a URN, *bp* is the URN of the business process for which the tuple is defined, *CR* is the set of authorisation credentials to be obtained from the Certificate and Credential Authority, *cca* to get authorised to invoke *bp*. *ape* is the URN of the Authorisation Policy Evaluator that requires these credentials. Each *bp* can have one or more of these (tuple) entries in the ARD.

Definition 17. BusinessProcess -Attribute-DAS tuple

We define BusinessProcess-Attribute-DAS tuple as *patd* = {*i, bp, AT, das, ape*}, where *i* is a URN, *bp* is the URN of the business process for which the tuple is defined, *AT* is the set of attributes to be obtained from a Dynamic Attribute Service, *das* to make an authorisation decision. *ape* is the URN of the Authorisation Policy Evaluator that requires these attributes. This means each *bp* can have one or more of these (tuple) entries in the ARD.

4.2.5 Credential Manager (CRM) Algorithms

CRM is an automated component that is invoked when a business process is created, or when an existing business process (definition) is modified or deleted. It is invoked to update the relevant runtime objects in the ARD. CRM runs different algorithms for static or pre-composed business processes and dynamic business processes.

4.2.5.1 Static Business Process Algorithm

The CRM algorithm for pre-composed or static business processes is defined as follows:
1. For every static business process BP_i defined in the BPAD,
   a. For each of the APEs involved,
      i. For each CCA responsible to send credentials to the APE, create a pcc tuple in the ARD
      ii. For each DAS the APE uses for runtime attributes, create a patd tuple in the ARD
   b. For each partner Web Service, locate the WS-Authorisation Policy statement (attached to its WSDL statement) and create a PartnerWebService element in the BP-AuthorisationPolicy (defined in Section 4.2.7.1) for BP_i
   c. For each partner business process, locate the BP-Authorisation Policy statement (attached to its WS-BPEL statement) and create a PartnerBusinessProcess element in the BP-AuthorisationPolicy for BP_i

4.2.5.2 Dynamic Business Process Algorithm

The CRM algorithm for dynamic business processes is defined as follows:
1. For every dynamic business process defined in the BPAD,
   a. For each of the APEs involved,
      i. For each CCA responsible to send credentials to the APE, create a pcc tuple in the ARD
      ii. For each DAS the APE uses for runtime attributes, create a patd tuple in the ARD

4.2.6 Authorisation Algorithms

Similar to WSAA, BPAA supports three authorisation algorithms. The first, *push-model* algorithm supports authorisations where a client's Client Proxy (CP), using the information in BP-AuthorisationPolicy, collects and sends the required credentials (from CCAs) and attributes (from DASs) to a Business Process Security Manager (BPSM). The second, *pull-model* algorithm supports authorisations where the Authorisation Server itself collects the required credentials from CCAs and APEs collect the required attributes from DASs. The AZS in this case uses the runtime objects' information from the ARD to be able to do so. The third, *combination-model* supports both the push and pull models of collecting the required credentials and attributes. An organisation must deploy one of these algorithms depending on the access control mechanisms used by their business processes.

When the combination-model algorithm is deployed by an organisation, the organisation's Authorisation Manager (AZM) may arbitrarily decide whether the credentials required from a CCA and dynamic attributes required from a DAS for each business process' APEs, are fetched by a CP (push-model) or by the authorisation components themselves (pull-model). The AZM may decide to give the entire responsibility of fetching the required credentials and attributes to the CP or to authorisation components or share the responsibility of fetching credentials and attributes amongst the client proxy and the authorisation components. This information is reflected in a business process' BP-AuthorisationPolicy (defined in Section 4.2.7.1). Appendix A shows the system sequence diagrams for push, pull and combination algorithms for both static and dynamic business processes. The sequence diagrams are explained in full detail in [14].

4.2.7 Extensions to the Description and Messaging Layers

Similar to WSAA, BPAA also require extensions to the Business Process Description and Messaging layers of SOA. We provide extensions to the SOAP header to carry authorisation

related credentials and attributes. We extend WS-BPEL (description layer) to include a business process authorisation policy (BP-AuthorisationPolicy) as well as the location of its BPSM.

4.2.7.1 BP-AuthorisationPolicy statement

WS-SecurityPolicy [35] is a statement consisting of a group of security policy "assertions", that represent a Web Service's security preference, requirement, capability or other property. We defined 'WS-AuthorisationPolicy' as a statement that contains a list of authorisation assertions, in Section 3.2.1. Similarly, we define a *BP-AuthorisationPolicy* here. That BP-AuthorisationPolicy includes assertions that specify what credentials (and from which CCA) and attributes (and from which DAS) a client's Client Proxy has to collect before invoking a business process. These assertions also include the credentials and attributes required to invoke a *static* business process' partner Web services as well as its partner business processes. We extend the WS-BPEL statement schema to include the BP-AuthorisationPolicy. Note that the partner Web services and business processes related authorisation information is not included in the BP-AuthorisationPolicy of a dynamic business process. Such information is only necessary for a static business process. Finally, the authorisation coordination information is also included in the BP-AuthorisationPolicy. This information is necessary only for dynamic business processes. Figure 12 shows the XML schema skeleton for the BP-AuthorisationPolicy.

```
<BP-AuthorisationPolicy>                          // For a Business Process, BPi
   <BusinessProcess>1
    <URN/>1                                       // BPi's URN
    <URN/>1                                       // BPi's URL
    <WS-BPEL-URL>1                                // Partner Business Process' WS-BPEL URL
    <Credentials>1..1                             // List of credentials required
       <CCA>1..*                                  // Certificate and Credential Authority (CCA)
          <CCA-URL/>1                             // Location of CCA
          <Credential/>1..*                       // Credentials provided by CCA
       </CCA>
    </Credentials>
    <Attributes>1..1                              // List of attributes required
       <DAS>1..*                                  // Dynamic Attribute Service (DAS)
          <DAS-URL/>1                             // Location of DAS
          <Attribute/>1..*                        // Attributes provided by DAS
       </DAS>
    </Attributes >
  </BusinessProcess>
                  //Credentials and attributes to invoke partner Web services
  <PartnerWebService>1..*
    <URN>1                                        // Partner Web service's URN
    <URL>1                                        // Partner Web service's URL
    <WSDL-URL>1                                   // Partner Web service's WSDL URL
    <Credentials>1..1                             // List of credentials required
       <CCA>1..*                                  // Certificate and Credential Authority (CCA)
          <CCA-URL/>1                             // Location of CCA
          <Credential/>1..*                       // Credentials provided by CCA
       </CCA>
    </Credentials>
    <Attributes>1..1                              // List of attributes required
       <DAS>1..*                                  // Dynamic Attribute Service (DAS)
          <DAS-URL/>1                             // Location of DAS
          <Attribute/>1..*                        // Attributes provided by DAS
       </DAS>
    </Attributes >
  </ PartnerWebService >
            //Credentials and attributes to invoke partner business processes
  <PartnerBusinessProcess>1..*
    <URN>1                                        // Partner Business Process' URN
    <URL>1                                        // Partner Business Process' URL
    <WS-BPEL-URL>1                                // Partner Business Process' WS-BPEL URL
    <Credentials>1..1                             // List of credentials required
       <CCA>1..*                                  // Certificate and Credential Authority (CCA)
          <CCA-URL/>1                             // Location of CCA
          <Credential/>1..*                       // Credentials provided by CCA
       </CCA>
    </Credentials>
    <Attributes>1..1                              // List of attributes required
       <DAS>1..*                                  // Dynamic Attribute Service (DAS)
          <DAS-URL/>1                             // Location of DAS
          <Attribute/>1..*                        // Attributes provided by DAS
       </DAS>
    </Attributes >
  </PartnerBusinessProcess >
                        // Authorisation Coordination information
  <AuthorisationCoordination>1..1
        <ActivationServiceLocation/>1..1          //Activation Service location
        <RegistrationServiceLocation/>1..1        //Registration Service location
        <AuthorisationCoordinatorLocation/>1..1   //Authorisation Coordinator location
  </AuthorisationCoordination>

</BP-AuthorisationPolicy>
```

Figure 12. XML schema skeleton for BP-AuthorisationPolicy (adopted from [14])

4.2.7.2 *Business Process Security Manager Location*

When a client wants to invoke a business process BP_1, its Client Proxy requires its Business Process Security Manager's location. We provide this information in BP_1's WS-BPEL statement. We introduce a new element *Business Process Security Manager* to the WS-BPEL statement. The XML schema skeleton for the Business Process Security Manager element and an example WS-BPEL statement are shown in Figure 13.

```
<businessProcessSecurityManager>  1..1
   <location>URL</location>
</businessProcessSecurityManager>
```

XML schema skeleton for Business Process Security Manager location

```
<process>
    <businessProcessSecurityManager>
         <location>http://www.travelagent.com</location>
    </businessProcessSecurityManager>

    <sequence>
        <receive partner="Customer"
            portType="purchaseOrderPT"
            operation="SendPurchaseOrder"
            container="PO">
        </receive>
        <invoke partner="CreditBureau"
            portType="CheckCreditPT"
            operation="CheckCredit">
        </invoke>
        <invoke partner="shippingProvider"
            portType="shippingPT"
            operation="RequestShipping"
            inputContainer="shipingRequest"
            outputContainer="shippingInfo">
                <source linkName="ship-to-invoice">
        </invoke>
        <reply partner="Customer"
            portType="purchaseOrderPT"
            operation="SendPurchaseOrder"
            container="Invoice"/>
    </sequence>
</process>
```

Figure 13. Extended WS-BPEL schema skeleton example (with BPSM location) (adopted from [14])

4.2.7.3 SOAP Header Extension

In Section 3, we extended the SOAP header to carry authorisation credentials and attributes required by a client to get authorised to a Web service. Similarly, we need a *SOAP-BP-AUTHZ* header to carry authorisation credentials and attributes required to access a business process. When a client wants to invoke a business process, its Client Proxy creates an authorisation header object and adds it to the SOAP Header before making a SOAP request to the business process. Note that the Client Proxy adds partner Web services' and business processes' related credentials and attributes only in the case of a static business process. We show the XML schema skeleton for the extended SOAP-BP-AUTHZ header in Figure 14.

```
<SOAP-ENV>
        <SOAP-ENV:Header>
                <SOAP-BP-AUTHZ>                          // For Business Process authorization
                    <BusinessProcess>1..1                // Credentials and attributes for business process
                        <Credentials>1..1                // List of collected credentials
                            <CCA>1..*                    // Certificate and Credential authority (CCA)
                                <CCA-URN/>1              // CCA identification
                                <Credential/>1..*        // Credentials provided by the CCA
                            </CCA>
                        </Credentials>
                        <Attributes>1..1                 // List of collected attributes
                            <DAS>1..*                    // Dynamic Attribute Service (DAS)
                                <DAS-URN/>1              // DAS identification
                                <Attribute/>1..*         // Attribute values provided by the DAS
                            </DAS>
                        </Attributes>
                    </BusinessProcess>
                    <PartnerWebService>1..*              // Credentials and attributes for partner WS
                        <URN>1                           // Partner Web service's URN
                        <Credentials>1..1                // List of collected credentials
                            <CCA>1..*                    // Certificate and Credential authority (CCA)
                                <CCA-URN/>1              // CCA identification
                                <Credential/>1..*        // Credentials provided by the CCA
                            </CCA>
                        </Credentials>
                        <Attributes>1..1                 // List of collected attributes
                            <DAS>1..*                    // Dynamic Attribute Service (DAS)
                                <DAS-URN/>1              // DAS identification
                                <Attribute/>1..*         // Attribute values provided by the DAS
                            </DAS>
                        </Attributes>
                    </PartnerWebService>
                    <PartnerBusinessProcess>1..*         // Credentials and attributes for partner BP
                        <URN>1                           // Partner business process' URN
                        <Credentials>1..1                // List of collected credentials
                            <CCA>1..*                    // Certificate and Credential authority (CCA)
                                <CCA-URN/>1              // CCA identification
                                <Credential/>1..*        // Credentials provided by the CCA
                            </CCA>
                        </Credentials>
                        <Attributes>1..1                 // List of collected attributes
                            <DAS>1..*                    // Dynamic Attribute Service (DAS)
                                <DAS-URN/>1              // DAS identification
                                <Attribute/>1..*         // Attribute values provided by the DAS
                            </DAS>
                        </Attributes>
                    </ PartnerBusinessProcess>
                </SOAP-BP-AUTHZ>
        </SOAP-ENV:Header>

        <SOAP-BODY>
                ...
        </SOAP-BODY>
</SOAP-ENV>
```

Figure 14. XML Schema skeleton for SOAP-BP-AUTHZ Header (adopted from [14])

4.3 Authorisation Coordination Framework for Dynamic Business Processes

We leverage the WS-Coordination [38] framework to coordinate authorisation messages between the Authorisation Coordinator and the participants (Client Proxy and Business Process Security Manager), to authorise a client to a dynamic business process' partner services. A dynamic business process binds to partner services dynamically at runtime. The coordination framework prescribes the following generic requirements:

- Activation of a new coordinator for the specific coordination protocol, for a particular application instance (in this case business process instance).
- Registration of participants with the coordinator, such that they receive coordination protocol messages when necessary.
- Propagation of coordination context information between coordination protocol participants.
- An entity to drive the coordination protocol through to completion.

The Client Proxy is responsible for the activation of a new instance of an Authorisation Coordinator. It is also the entity responsible to drive the coordination protocol to completion. It is aware that authorisation coordination is required to get the client authorised to a dynamic business process, as the BP-AuthorisationPolicy has the information about the Authorisation Coordinator, the coordination protocol used and its type (authorisation coordination), and finally its location.

The Business Process Security Manager is another participant in the coordination protocol. During the course of execution of a dynamic business process, if the WS-BPEL Engine (BPEL Engine for short) needs to invoke a partner service, it sends a message about the same to the business process' Business Process Security Manager. The Business Process Security Manager then informs the Authorisation Coordinator that a partner service has been invoked and it needs authorisation credentials from the client (Client Proxy). The Authorisation Coordinator informs the Client Proxy about the same. The Client Proxy then fetches the required credentials and gets back to the Authorisation Coordinator. The Authorisation Coordinator then sends a message with the received credentials to the Business Process Security Manager. The Business Process Security Manager sends these credentials to the BPEL Engine. The BPEL engine uses these credentials and then continues execution of the partner service.

Figure 15 shows the initial stages of invoking a dynamic business process. The Client Proxy locates the activation service and sends it a message asking for the creation of an Authorisation Coordinator and a corresponding coordination context. An example of a *CreateCoordinationContext* message is shown in Figure 16. The activation service's location, the requester reference (Client Proxy's address) as well as the URI of the coordination type (authorisation coordination) are included in the activation message.

Assuming that an authorisation coordination service has been registered with the coordination framework, a coordinator is created (and exposed as a registration service) and a context such as that shown in Figure 17 is duly returned to the Client Proxy as part of the *CreateCoordinationContextResponse* message. We have also included the business process' URN (in the place of *any* element in the message's XML schema definition) in the coordination context response message to propagate the identity of the business process for which authorisation messages are coordinated. It is shown in bold in Figure 17.

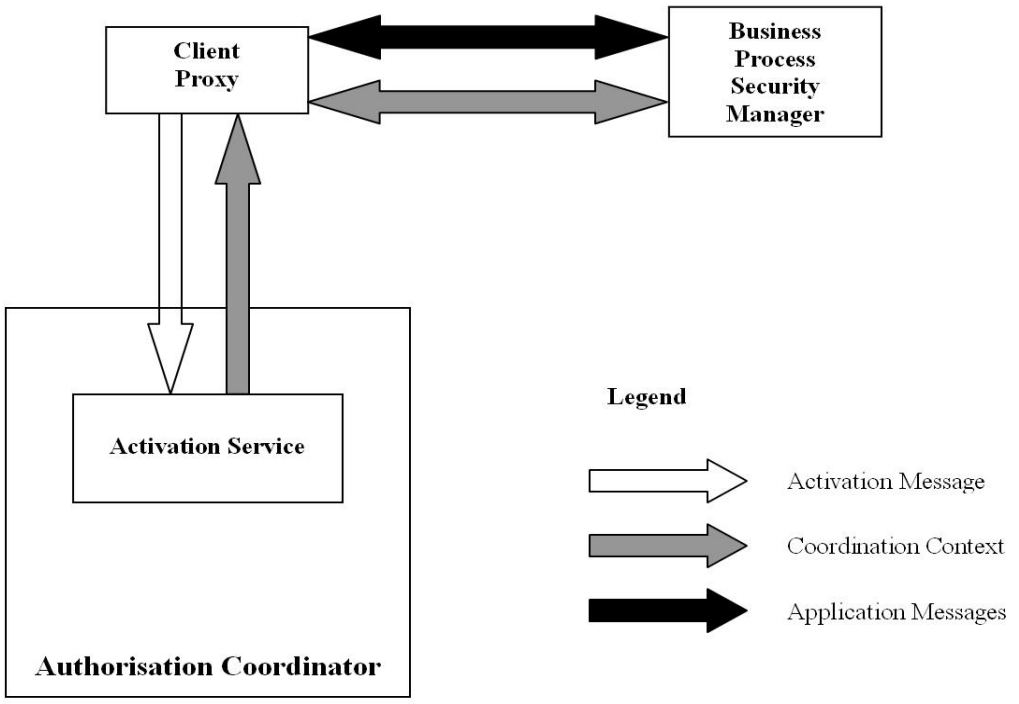

Figure 15. Authorisation Coordination Framework: Activation and Application Messages (adopted from [14])

```
<soap:Envelope
        xmlns:soap="http://www.w3.org/2002/06/soap-envelope">
   <soap:Body>
        <CreateCoordinationContext
                xmlns="http://schemas.xmlsoap.org/ws/2002/08/wscoor"
                xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
                <ActivationService>
                        <wsu:Address>
                                http://bogey.ics.mq.edu.au/authzcoord/activationservice.asmx
                        </wsu:Address>
                </ActivationService>
                <RequesterReference>
                        <wsu:Address>
                                http://workstation.mq.edu.au/station11/ClientProxy.asmx
                        </wsu:Address>
                </RequesterReference>
                <CoordinationType>
                        http://bogey.ics.mq.edu.au/authzcoord
                </CoordinationType>
        </CreateCoordinationContext>
   </soap:Body>
</soap:Envelope>
```

Figure 16. Authorisation Coordination Activation Message (adopted from [14])

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
  <S:Header>
    . . .
    <wscoor:CoordinationContext
      xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
      xmlns:wscoor=http://schemas.xmlsoap.org/ws/2002/08/wscoor
      xmlns:bp="http:bogey.ics .mq.edu.au/businessprocess"
      <wsu:Expires>
        2002-06-30T13:20:00.000-05:00
      </wsu:Expires>
      <wsu:Identifier>
        http://AuthorizationCoordination.asmx/v1.0
      </wsu:Identifier>
      <wscoor:CoordinationType>
        http:// bogey.ics.mq.edu.au/authzcoord
      </wscoor:CoordinationType>
      <wscoor:RegistrationService>
        <wsu:Address>
            http://bogey.ics.mq.edu.au/authzcoord/registrationservice.asmx
        </wsu:Address>
        <bp:BusinessProcessURN>http:bogey.ics.mq.edu.au/BP1.bpel</bp:BusinessProcessURN>
      </wscoor:RegistrationService>
    </wscoor:CoordinationContext>
    . . .
  </S:Header>
```

Figure 17. Coordination Context example (adopted from [14])

The Client Proxy interacts with the Business Proxy Security Manager sending and receiving messages as normal, with the exception that it embeds the authorisation coordination context (which carries the authorisation information) in a SOAP header block in its messages to provide authorisation related credentials for those partner services (Web services and/or business processes) that are invoked. Also the Client Proxy itself registers as a participant with the Authorisation Coordinator.

The Business Process Security Manager understands the protocol messages associated with our authorisation service. If it has not registered a participant previously, it does so once it receives a SOAP message from the Client Proxy, containing an authorisation context header, using the details provided in the context (via the WS-Coordination registration service URI). This register operation occurs every time that the Business Process Security Manager receives a particular context for the first time.

When the Client Proxy receives the final response from the Business Process Security Manager after execution of the business process, it sends a *Completion Message* to the Authorisation Coordinator. The Authorisation Coordinator then sends the Completion Message to the Business Process Security Manager, registered as a participant to the Authorisation Coordinator. Any subsequent calls by the Client Proxy (on behalf of the client) to that business process with the same context will result in the service being unable to register a participant since the context details will no longer resolve to a live coordinator to register with. This second part of the lifecycle of the authorisation coordination is shown in Figure 18. An example of *Register* and *RegisterResponse* messages is shown in Figure 19. A register message consists of the address of the registration service, the address of the requester (Client Proxy), protocol identifier (authorisation coordination protocol), and finally the participant's (Client Proxy Participant WSDL interface's) address. A register response message comprises the requester address (Client Proxy's address) and the coordination protocol service's address (*Authorisation Service*'s address, see Figure 19).

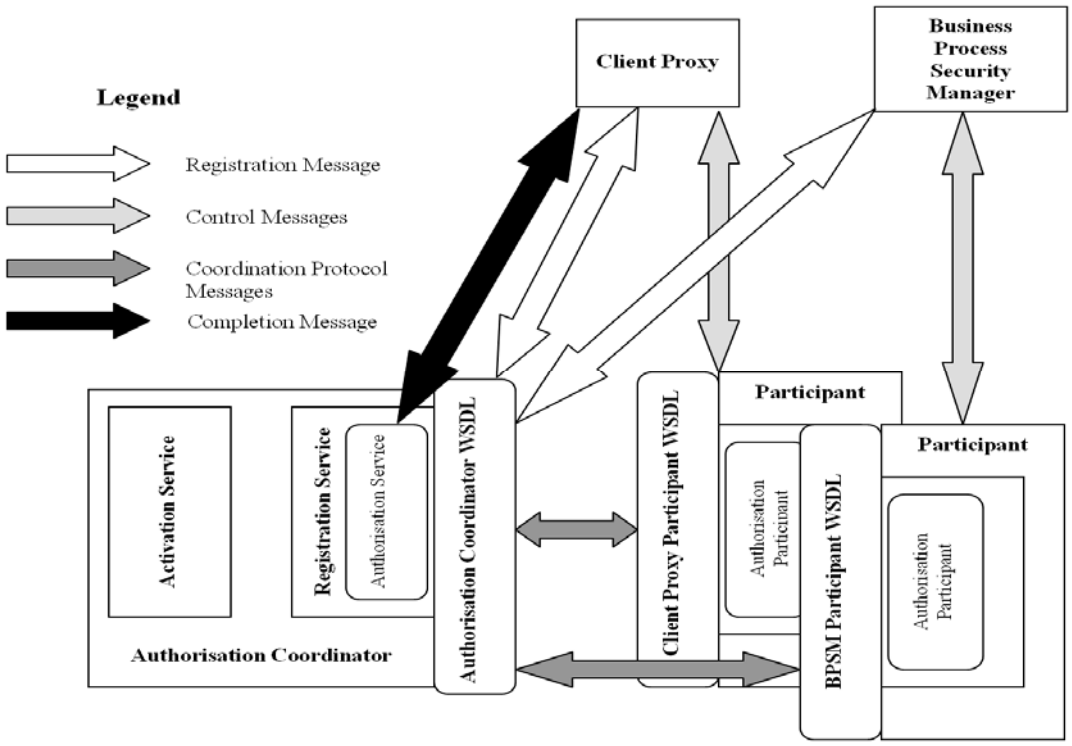

Figure 18. Authorisation Coordination Framework: Registration and Coordination Messages (adopted from [14])

The authorisation coordination messages passed between the Authorisation Coordinator and the participants are asynchronous. Therefore, we need to define standard interfaces both at the coordinator end – for the Activation Service, the Registration Service and the Authorisation Service (see Figure 18), as well as for the participant ends — for Client Proxy Participant and Business Process Security Manager Participant.

4.3.1 Authorisation Coordinator WSDL Interfaces

Activation Service and Registration Service WSDL interfaces are exactly the same as defined in the WS-Coordination framework. We show them in Figure 20. The Authorisation Service handles the authorisation coordination protocol related messages. We define the WSDL interface for the Authorisation Service in Figure 21. The interface exposes four different asynchronous operations. The *GetPartnerCredentials* operation is invoked by the Business Process Security Manager, when credentials are required by the BPEL Engine to invoke partner services. The Client Proxy invokes the overloaded *PartnerCredentials* operation to send the authorisation credentials and attributes required to invoke a partner service (parameter of type authzcoor:CredentialsWSResponse is used to send a partner Web service's credentials; parameter of type authzcoor:CredentialsBPResponse is used to send a partner business process' credentials). The Client Proxy invokes the *CompletionMessage* operation to inform the Authorisation Coordinator to terminate the coordination process.

An example of Register Message

```
<Register>
  <RegistrationService>
    <wsu:Address>
        http://bogey.ics.mq.edu.au/authzcoord/registrationservice.asmx
    </wsu:Address>
  </RegistrationService>
  <RequesterReference>
    <wsu:Address>
       http://workstation.mq.edu.au/station11/CientProxy.asmx
    </wsu:Address>
  </RequesterReference>
  <ProtocolIdentifier>
    http:// bogey.ics.mq.edu.au/authzcoord/authzprotocol
  </ProtocolIdentifier>
  <ParticipantProtocolService>
    <wsu:Address>
        http://workstation.mq.edu.au/station11/ClientProxyParticipant.asmx
    </wsu:Address>
   </ParticipantProtocolService>
</Register>
```

An example of Register Response Message

```
<RegisterResponse>
   <RequesterReference>
     <wsu:Address>
        http://workstation.mq.edu.au/station11/CientProxy.asmx
      </wsu:Address>
   </RequesterReference>
  <CoordinatorProtocolService>
    <wsu:Address>
        http://bogey.ics.mq.edu.au/authorizationservice.asmx/
    </wsu:Address>
  </CoordinatorProtocolService>
</RegisterResponse>
```

Figure 19. Register Request and Response Messages (adopted from [14])

```
<!-- Activation Service portType Declaration -->
<wsdl:portType name="ActivationCoordinatorPortType">
        <wsdl:operation name="CreateCoordinationContext">
                <wsdl:input
                        message="wscoor:CreateCoordinationContext"/>
        </wsdl:operation>
</wsdl:portType>

<!-- Registration Service portType Declaration -->
<wsdl:portType name="RegistrationCoordinatorPortType">
        <wsdl:operation name="Register">
                <wsdl:input message="wscoor:Register"/>
        </wsdl:operation>
</wsdl:portType>
```

Figure 20. Activation and Registration Service WSDL interfaces (adopted from [14])

```
<!—Authorisation Service portType Declaration -->

<wsdl:portType name="AuthorisationServicePortType">
        <wsdl:operation name="GetPartnerCredentials">
                <wsdl:input message="authzcoor:CredentialsRequest"/>
        </wsdl:operation>
        <wsdl:operation name="PartnerCredentials">
                <wsdl:input
        message="authzcoor:CredentialsWSResponse"/>
        </wsdl:operation>
        <wsdl:operation name="PartnerCredentials">
                <wsdl:input
        message="authzcoor:CredentialsBPResponse"/>
        </wsdl:operation>
        <wsdl:operation name="CompletionMessage"/>
</wsdl:portType>
```

Figure 21. Authorisation Service WSDL interface (adopted from [14])

4.3.2 Participant Interfaces

We show the Client Proxy Participant WSDL interface in Figure 22. The interface exposes four operations. The *CreateCoordinationContextResponse* operation is invoked by the Authorisation Coordinator (Activation Service) to send the coordination context to the Client Proxy. The *RegisterResponse* operation is invoked by the Authorisation Coordinator (Registration Service) to send the registration response message to the Client Proxy. The *GetPartnerCredentials* is invoked by the Authorisation Coordinator (Authorisation Service) to inform the Client Proxy that authorisation credentials are required to invoke a partner service (Web service or business process). The Authorisation Coordinator (Authorisation Service) invokes the *Error* operation to let the Client Proxy know of any coordination related error.

```
<!—Client Proxy Participant portType Declaration -->

<wsdl:portType name="ClientProxyParticipantPortType">
        <wsdl:operation
                name="CreateCoordinationContextResponse">
                <wsdl:input
                  message="wscoor:CreateCoordinationContextResponse"/>
        </wsdl:operation>
        <wsdl:operation name="RegisterResponse">
                <wsdl:input message="wscoor:RegisterResponse"/>
        </wsdl:operation>
        <wsdl:operation name="GetPartnerCredentials">
                <wsdl:input message="authzcoor:CredentialsRequest"/>
        </wsdl:operation>
        <wsdl:operation name="Error">
                <wsdl:input message="wscoor:Error"/>
        </wsdl:operation>
</wsdl:portType>
```

Figure 22. Client Proxy Participant WSDL Interface (adopted from [14])

We show the Business Process Security Manager Participant WSDL interface in Figure 23. The interface exposes four operations. The *CreateCoordinationContextResponse* operation is invoked by the Authorisation Coordinator (Activation Service) to send the coordination context to the Business Process Security Manager. The *RegisterResponse* operation is invoked by the Authorisation Coordinator (Registration Service) to send the registration response message to the Business Process Security Manager. The *GetPartnerCredentials* is invoked by the BPEL Engine (responsible to execute a business process) to inform the Business Process Security Manager that authorisation credentials are required to invoke a partner service (Web service or business process). The Authorisation Coordinator (Authorisation Service) invokes the overloaded *PartnerCredentials* operation to send the authorisation credentials and attributes required to invoke a partner service (parameter of type authzcoor:CredentialsWSResponse is used to send a partner Web service's credentials; parameter of type authzcoor:CredentialsBPResponse is used to send a partner business process' credentials). The Authorisation Coordinator (Authorisation Service) invokes the *Error* operation to let the Business Process Security Manager know of any coordination related error.

```
<!—Business Process Security Manager Participant portType Declaration -->

<wsdl:portType name=" BPSMParticipant PortType">
        <wsdl:operation
                name="CreateCoordinationContextResponse">
                <wsdl:input
                   message="wscoor:CreateCoordinationContextResponse"/>
        </wsdl:operation>
        <wsdl:operation name="RegisterResponse">
                <wsdl:input message="wscoor:RegisterResponse"/>
        </wsdl:operation>
        <wsdl:operation name="GetPartnerCredentials">
                <wsdl:input message="authzcoor: CredentialsRequest"/>
        </wsdl:operation>
        <wsdl:operation name="PartnerCredentials">
                <wsdl:input message="authzcoor: CredentialsWSResponse"/>
        </wsdl:operation>
        <wsdl:operation name="PartnerCredentials">
                <wsdl:input message="authzcoor: CredentialsBPResponse"/>
        </wsdl:operation>
        <wsdl:operation name="Error">
                <wsdl:input message="wscoor:Error"/>
        </wsdl:operation>
</wsdl:portType>
```

Figure 23. Business Process Security Manager Participant WSDL Interface (adopted from [14])

Figures A-4, A-5, and A-6 in Appendix A show the system sequence diagrams for dynamic business process authorisation using the coordination framework.

4.4 Extension to the Authorisation Coordination Framework

At the beginning of Section 4, we assumed that a dynamic business process has partner services that are either Web services or *static* business processes. In this section, we extend the authorisation coordination framework to allow a dynamic business process to invoke other dynamic business processes. In other words, a dynamic business process may have another dynamic business process as a partner.

We extend the WSDL interfaces of the Business Process Security Manager Participant and the Authorisation Service (part of the Authorisation Coordinator) to carry the required authorisation coordination related messages amongst them. We show these extended WSDL interfaces in Figures 24 and 25.

Let us consider a scenario, where a dynamic business process, BP_1, invokes another dynamic business process, BP_2.

We introduce an asynchronous operation, *InvokeDynamicBP* (shown in bold in Figure 24), to the Business Process Participant WSDL interface. It is invoked by the BPEL Engine executing BP_1 to inform BP_1's Business Process Security Manager that a partner dynamic business process (BP_2) has been invoked by BP_1.

```
<!—Business Process Security Manager Participant portType Declaration -->

<wsdl:portType name=" BPSMParticipant PortType">
        <wsdl:operation
                name="CreateCoordinationContextResponse">
                <wsdl:input
                  message="wscoor:CreateCoordinationContextResponse"/>
        </wsdl:operation>
        <wsdl:operation name="RegisterResponse">
                <wsdl:input message="wscoor:RegisterResponse"/>
        </wsdl:operation>
        <wsdl:operation name="GetPartnerCredentials">
                <wsdl:input message="authzcoor:CredentialsRequest"/>
        </wsdl:operation>
        <wsdl:operation name="PartnerCredentials">
                <wsdl:input message="authzcoor:CredentialsWSResponse"/>
        </wsdl:operation>
        <wsdl:operation name="PartnerCredentials">
                <wsdl:input message="authzcoor:CredentialsBPResponse"/>
        </wsdl:operation>
        <wsdl:operation name="InvokeDynamicBP" />
        <wsdl:operation name="Error">
                <wsdl:input message="wscoor:Error"/>
        </wsdl:operation>
</wsdl:portType>
```

Figure 24. Extended Business Process Security Manager WSDL Interface (adopted from [14])

We introduce an asynchronous operation, *InvokeDynamicBP*, to the Authorisation Service WSDL interface. It is invoked by the Business Process Security Manager (BP_1's in this case) to inform BP_1's Authorisation Coordinator, that a partner dynamic business process (in this case BP_2) has been invoked. Now BP_1's Authorisation Coordinator acts as a participant with BP_2's Authorisation Coordinator. Therefore we need the Authorisation Service WSDL interface to also expose the *ActivationResponse* and *RegistrationResponse* (asynchronous) operations, to receive the activation and registration response messages from partner business process' Authorisation Coordinator (in this case BP_2's). The newly introduced operations are shown in bold in Figure 25.

We now discuss the sequence of steps involved in authorisation coordination when a dynamic business process (BP_1) invokes another dynamic business process (BP_2). Figure 26 shows the

sequence diagram. For clarity of discussion, we only show the authorisation coordination framework related steps in Figure 26. The other steps involved in authorisation algorithms, such as collection of credentials and attributes by the Client Proxy, and authorisation of the client by authorisation components, are not shown in the figure.

```
<!—Authorisation Service portType Declaration -->

<wsdl:portType name="AuthorisationServicePortType">
        <wsdl:operation name="GetPartnerCredentials">
                <wsdl:input message="authzcoor:CredentialsRequest"/>
        </wsdl:operation>
        <wsdl:operation name="PartnerCredentials">
                <wsdl:input message="authzcoor:CredentialsWSResponse"/>
        </wsdl:operation>
        <wsdl:operation name="PartnerCredentials">
                <wsdl:input message="authzcoor:CredentialsBPResponse"/>
        </wsdl:operation>
        <wsdl:operation name="InvokeDynamicBP"/>
        <wsdl:operation
                name="CreateCoordinationContextResponse">
                <wsdl:input
                  message="wscoor:CreateCoordinationContextResponse"/>
        </wsdl:operation>
        <wsdl:operation name="RegisterResponse">
                <wsdl:input message="wscoor:RegisterResponse"/>
        </wsdl:operation>
        <wsdl:operation name="CompletionMessage"/>
</wsdl:portType>
```

Figure 25. Extended Authorisation Service WSDL Interface (adopted from [14])

4.4.1 Extended Authorisation Coordination Steps

(1) The Client Proxy sends an asynchronous activation message to BP_1's Authorisation Coordinator (ACO-BP_1).
(2) ACO-BP_1 sends an asynchronous activation response message to the Client Proxy.
(3) The Client Proxy sends an asynchronous register message to ACO-BP_1
(4) ACO-BP_1 sends an asynchronous registration response message to the Client Proxy.
(5) The Client Proxy sends an asynchronous invocation request to BP_1's Business Process Security Manager (BPSM-BP_1), along with the authorisation coordination context in SOAP header.
(6) BPSM-BP_1 sends an asynchronous register message to ACO-BP_1.
(7) ACO-BP_1 sends an asynchronous registration response message to BPSM-BP_1.
(8) The Business Process Security Manager acts as a broker for the client's request and sends the (asynchronous) invocation message to BP_1's BPEL Engine (BPEL-BP_1).
(9) BPEL-BP_1 starts executing BP_1. In the course of BP_1's execution, when BP_2, a dynamic business process needs to be invoked, BPEL-BP_1 stops execution.
(10) BPEL-BP_1 sends an asynchronous (using *InvokeDynamicBP* operation) to BPSM-BP_1.
(11) BPSM-BP_1 uses BP_2's WS-BPEL statement and sends an asynchronous activation message to BP_2's Authorisation Coordinator (ACO-BP_2).
(12) ACO-BP_2 sends an asynchronous activation response message to the BPSM-BP_1, along with the authorisation coordination context (Context-BP_2).

(13) BPSM-BP_1 sends an asynchronous message (using *InvokeDynamicBP* operation) along with the Context-BP_2 in the SOAP header to the ACO-BP_1.
(14) ACO-BP_1 using the information in Context-BP_2 sends an asynchronous register message to ACO-BP_2.
(15) ACO-BP_2 sends an asynchronous registration response message to ACO-BP_1.
(16) ACO-BP_1 sends an asynchronous message (using *InvokeDynamicBP* operation) along with the Context-BP_2 in the SOAP header, to BPSM-BP_1.
(17) BPSM-BP_1 sends an asynchronous message (using *InvokeDynamicBP* operation) to BPEL-BP_1.
(18) BPEL-BP_1 invokes BP_2. In the course of execution of BP_2, when a partner Web service or static business process is invoked, its BPEL Engine (BPEL-BP_2) informs its Business Process Security Manager (BPSM-BP_2), which in turn informs ACO-BP_2. All these steps happen at this stage and are not shown in Figure 26.
(19) ACO-BP_2 sends an asynchronous message to ACO-BP_1 (using *GetPartnerCredentials* operation). Note that ACO-BP_1 is a registered as a participant with ACO-BP_2.
(20) ACO-BP_1 sends an asynchronous message to Client Proxy (using *GetPartnerCredentials* operation on Client Proxy Participant interface).
(21) The Client Proxy uses the WSDL or WS-BPEL statement of the partner service and collects the required authorisation credentials and attributes (steps not shown in Figure 26) and sends them back to ACO-BP_1 using an asynchronous operation (*PartnerCredentials*).
(22) ACO-BP_1 sends the credentials and attributes to ACO-BP_2 using an asynchronous operation (*PartnerCredentials*). ACO-BP_2 sends them to BPSM-BP_2, which in turn sends them to BPEL-BP_2 (not shown in Figure 26).
(23) BPEL-BP_2 continues with the execution of BP_2 and gets back to BPEL-BP_1 (not shown in Figure 26). BPEL-BP_1 sends BP_1's invocation result to BPSM-BP_1 as an asynchronous message.
(24) BPSM-BP_1 sends BP_1's invocation result as an asynchronous message to the Client Proxy after the complete execution of BP_1. The Client Proxy sends the result to the client interface.

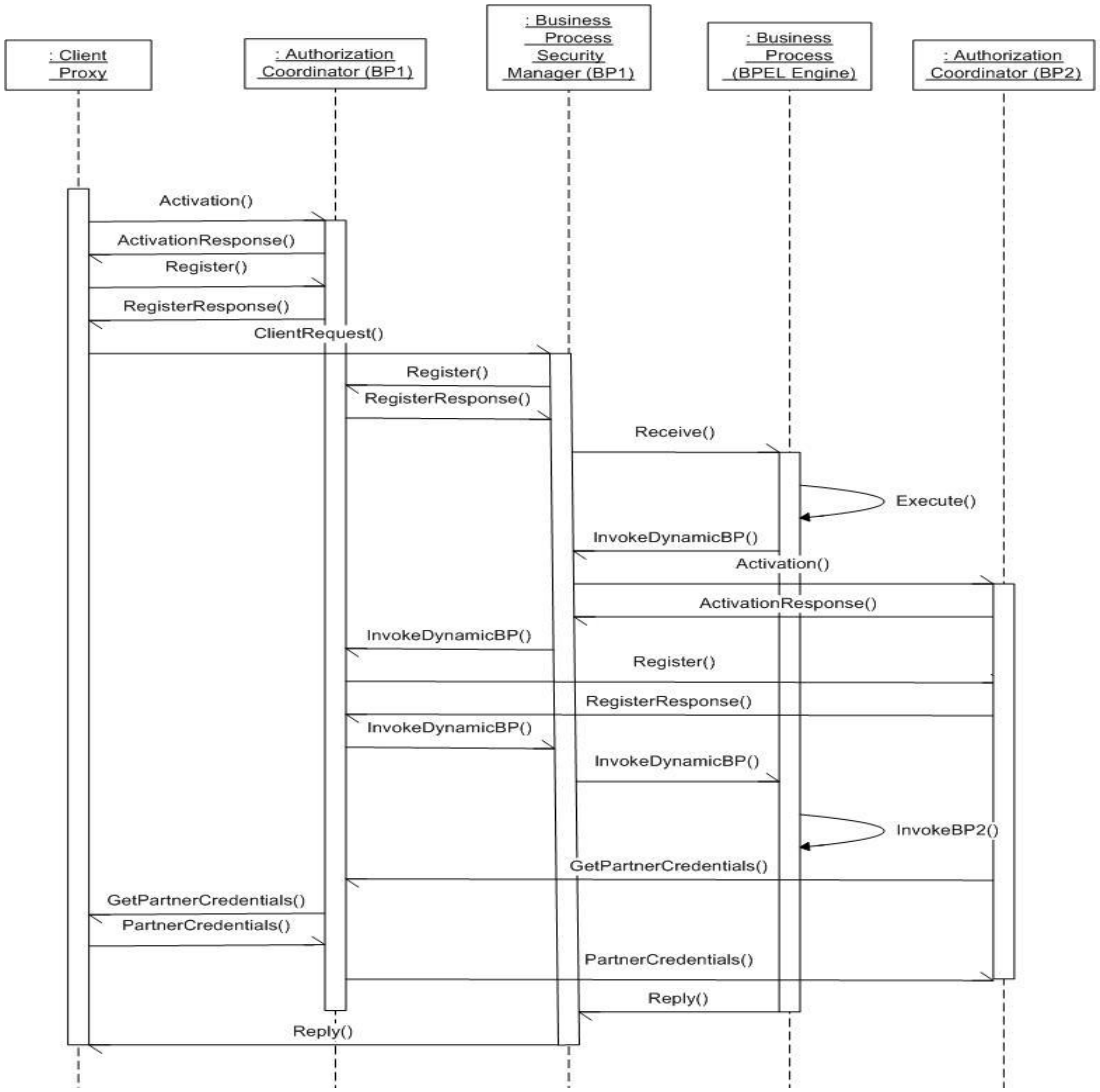

Figure 26. Extended Authorisation Coordination Framework (adopted from [14])

4.4.2 Discussion

We have discussed how BPAA handles client authorisation to both static and dynamic business processes. We initially assumed that their partner business processes are always static. In Section 4.4.1, we extended the authorisation framework to handle authorisation where a dynamic business process invokes another partner dynamic business process. However, there is another scenario we have not covered. What happens when a *static* business process, say BP_1, invokes another partner that is a *dynamic* business process, say BP_2?

In this case, the BPEL Engine invoking BP_1, informs BP_1's Business Process Security Manager (BPSM) that a partner (dynamic) business process, BP_2, has been invoked. The Business Process Security Manager sends a message to the Client Proxy about the same, along with the location of the WS-BPEL statement of BP_2. The Client Proxy activates BP_2's Authorisation Coordinator (ACO-BP_2), registers as a participant, and then participates in the authorisation coordination protocol with (ACO-BP_2) and gets the client authorised to BP_2. Once BP_2 finishes execution, it returns to BP_1. BP_1's BPEL Engine notifies its BPSM (BPSM-BP_1) that BP_2 has returned with a response, and then continues BP_1's execution. BPSM-BP_1 informs the Client Proxy that BP_2 has returned with a response. Then the Client Proxy sends a completion message to ACO-BP_2 and ends the authorisation coordination session with ACO-BP_2.

4.5 Benefits of the BPAA

The BPAA provides all the benefits of the WSAA (see Section 3.3) as well as the following benefits:

(a) Support for both static and dynamic business processes: A business process can either be static, where it is pre-composed and the partner services are known at design time, or, dynamic, where only the interfaces to the partner services are exposed by the business processes, and the binding to real services is made at runtime depending on client requirements. Also a business process may have Web services or even other business processes as partners. We took all such scenarios into consideration and provided a comprehensive architecture for authorisation for the business processes layer of SOA. We provided a comprehensive authorisation coordination framework to authorise clients to dynamic business processes. Also we extended our authorisation coordination framework to allow for both static and dynamic business processes to invoke partner services that are themselves dynamic business processes.

(c) Decentralised security administration: The partners involved in a business process workflow are allowed to autonomously control their authorisation policies. The partners can be either from within an organisation or from multiple organisations. In the case of static business processes, the information about the authorisation credentials required to invoke partner services is exposed in the WS-BPEL statement (using BP-AuthorisationPolicy) at design time itself.

(d) Dynamic discovery and orchestration of a business process' partners' authorisation evaluation components: In the case of dynamic business processes, the BPAA coordinates the authorisation where binding to real partner services happens at runtime depending on client requirements. The Authorisation Coordination components' location is exposed to the client (Client Proxy) using the BP-AuthorisationPolicy attached to the WS-BPEL statement. When the business process workflow reaches a stage where some credentials are required by the access control system of the partner involved, the Authorisation Coordinator makes the Client Proxy aware of this. The Client Proxy fetches the required credentials using the partner service WSDL or WS-BPEL statement and sends them to the Authorisation Coordinator. The Authorisation Coordinator sends them to the Business Process Security Manager, a participant in the coordination protocol, which in turn sends it to the BPEL Engine to continue invocation of the business process.

(e) Non-disclosure of policies: The BPAA does not require the partner services to disclose their policies to the partner that is controlling the business process. The authorisation of the client happens at the same place, where the partners originally intended it to be. For example, if a Travel Agent Service (TAS) creates a business process that binds and interacts with Qantas airlines, Hertz car rental and Hilton hotel at design or run time, the TAS does not require the different partners involved to disclose their policies in order to manage the authorisation decisions involved. The authorisation of the client to these partner services is evaluated based on their own policies and at their own policy decision points. Therefore, organisations can now leverage the services offered by the BPAA, and do business by binding to portal agents even if they do not trust them to perform client authorisation.

5. Concluding Remarks

We proposed a comprehensive authorisation framework for SOA in this report. The framework comprises the WSAA and the BPAA. We described the architectural framework, the

administration and runtime aspects of both our architectures and their components for secure authorisation of Web services and business processes as well as the support for the management of authorisation information. Both the WSAA and the BPAA support push-model, pull-model and combination-model authorisation algorithms.

The WSAA has been implemented and integrated it into the Microsoft .NET framework and the applicability of the WSAA to the healthcare domain has been demonstrated in [29]. As part of future work, we would also like to implement the BPAA and demonstrate our comprehensive SOA authorisation framework using an appropriate Defence application.

Our understanding is that IBM's Defence Operations Platform (DOP)[3] has been adopted by the Australian Defence to build SOA services. We believe there is scope in the future to take into consideration real Defence services and their access control requirements, and research potential integration points of our comprehensive SOA authorisation framework into the DOP.

6. References

1. Rescorla, E. (2001) *SSL and TLS: Designing and Building Secure Systems*, Addison Wesley
2. IETF Secretariat (2004) *IP Security Protocol,* http://www.ietf.org/html.charters/ipsec-charter.html.
3. World Wide Web Consortium (W3C) (2008) *XML-Signature Syntax and Processing,* http://www.w3.org/TR/xmldsig-core/.
4. World Wide Web Consortium (W3C) (2002) *XML Encryption Syntax and Processing,* http://www.w3.org/TR/xmlenc-core.
5. World Wide Web Consortium (W3C) (2003) *SOAP v1.2,* http://www.w3.org/TR/soap12-part1/.
6. World Wide Web Consortium (W3C). *Web Services Description Language (WSDL) v1.1,* http://www.w3.org/TR/wsdl. (2001) [Accessed.
7. B. Atkinson et al (2002) Web Services Security (WS-Security) Specification, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/. April
8. S. Anderson et al. (2005) Web Services Trust Language (WS-Trust), http://www-106.ibm.com/developerworks/library/specification/ws-trust/.
9. Varadharajan, V. (2002) Distributed Authorization: Principles and Practice. In: *Coding Theory and Cryptology, Lecture Notes Series, Institute for Mathematical Sciences, National University of Singapore*. Singapore University Press
10. Agarwal, S., Sprick, B. and Wortmann, S. (2004) Credential Based Access Control for Semantic Web Services. *American Association for Artificial Intelligence*
11. Kraft, R. (2002) Designing a Distributed Access Control Processor for Network Services on the Web. In: *ACM Workshop on XML Security,* Fairfax, VA, USA: November 22
12. Yagüe, M. I. and Troya, J. M. (2002) *Euroweb 2002 Conference. The Web and the GRID: from e-science to e-business*, Oxford, UK
13. Ziebermayr, T. and Probst, S. (2004) *International Conference on Web Services (ICWS)*, San Diego, CA, USA
14. Indrakanti, S. (2007) *PhD Thesis: On Engineering Authorization Systems for Web Services based Service-Oriented Architecture*. Sydney, Macquarie University
15. OASIS (2007) *Web Services Business Process Execution Language Version 2.0,* http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
16. Sandhu, R., et al. (1996) Role-Based Access Control Models. *IEEE Computer* **29(2)** 38-47

[3] http://www-01.ibm.com/software/industry/defense-operations-platform/

17. Oracle. *JAAS Authorization Tutorial*. (2000) [Accessed 20 April, 2011]; Available from: http://download.oracle.com/javase/1.5.0/docs/guide/security/jaas/tutorials/index.html.
18. Thomas S. Cook, et al. *Orchestrating BMD Control in Extended BPEL*. Naval postgraduate School
19. Duminda Wijesekera, James B. Michael and Anil Nerode (2005) BMD Agents: An Agent-Based Framework to Model Ballistic Missile Defense Strategies. In: *6th International Workshop on Policies for Distributed Systems and Networks, IEEE* Stockholm, Sweden
20. Karp, A. H. (2006) Authorization-Based Access Control for the Service Oriented Architecture. In: *Fourth International Conference on Creating, Connecting, and Collaborating through Computing,* Berkeley, CA, USA: 26-27 January 2006
21. Indrakanti, S. and Varadharajan, V. (2005) An Authorization Architecture for Web Services. In: *19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security,* Storrs, Connecticut, USA, Springer LNCS
22. Koshutanski, H. and Massacci, F. (2002) *An Access Control System for Business Processes for Web Services*. DIT-02-102, [Technical Report] Informatica e Telecomunicazioni, University of Trento
23. Mont, M. C., Baldwin, A. and Pato, J. (2003) *Secure Hardware-based Distributed Authorisation Underpinning a Web Service Framework*. HPL-2003-144,
24. Bertino, E., Crampton, J. and Paci, F. (2006) Access Control and Authorization Constraints for WS-BPEL. In: *International Conference on Web Services (ICWS)*: 18-22 Sept. 2006
25. Indrakanti, S. and Varadharajan, V. (2011) Coordination based Distributed Authorization for Business Processes in Service Oriented Architectures. In: *The Sixth International Conference on Internet and Web Applications and Services,* St. Maarten, The Netherlands Antilles
26. Indrakanti, S. (2012) *On the Design Requirements for a Comprehensive SOA Authorisation Framework; DSTO-CR-2011-0251* DSTO
27. Indrakanti, S., Varadharajan, V. and Hitchens, M. (2005) Analysis of Existing Authorization Models and Requirements for Design of Authorization Framework for the Service Oriented Architecture. In: *The 2005 International Symposium of Web Services and Applications,* Las Vegas, USA: June 27-30
28. Indrakanti, S., Varadharajan, V. and Hitchens, M. (2005) Principles for the Design of Authorization Framework for the Service Oriented Architecture. In: *International Conference on Internet Technologies and Applications (ITA 05),* Wrexham, North Wales, UK: September 7-9
29. Indrakanti, S., Varadharajan, V. and Agarwal, R. (2007) On the design, implementation and application of an authorisation architecture for web services. *International Journal of Information and Computer Security* **1** (1/2)
30. Bellwood, T., et al. *UDDI Specification version 3.0.2, http://uddi.org/pubs/uddi_v3.htm*. (2004) [Accessed.
31. ITU-T Recommendation, June 1997. 459 (1997) *X.509 (1997 E): Information Technology - Open Systems Interconnection - The Directory: Authentication Framework*.
32. Bacon, J. and Moody, K. (2002) Toward open, secure, widely distributed services. *Communications of the ACM* **45** (6) 59-64
33. Chadwick, D. W. and Otenko, A. (2002) The PERMIS X.509 role based privilege management infrastructure. In: *Proceedings of the seventh ACM symposium on Access control models and technologies*. ACM Press 135-140
34. Kraft, R. (2002) A model for network services on the web. In: *The 3rd International Conference on Internet Computing (IC 2002)*: June

35. Giovanni Della-Libera et al. (2002) Web Services Security Policy Language (WS-SecurityPolicy), http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-securitypolicy.asp.
36. S. Bajaj et al (2004) Web Services Policy Attachment (WS-PolicyAttachment), http://www-106.ibm.com/developerworks/library/specification/ws-polatt/. September
37. B. Atkinson et al (2002) WS-Security Specification, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/. April
38. OASIS (2009) *Web Services Coordination (WS-Coordination)*.

Appendix A

1. Sequence diagrams for static business process authorisation algorithm

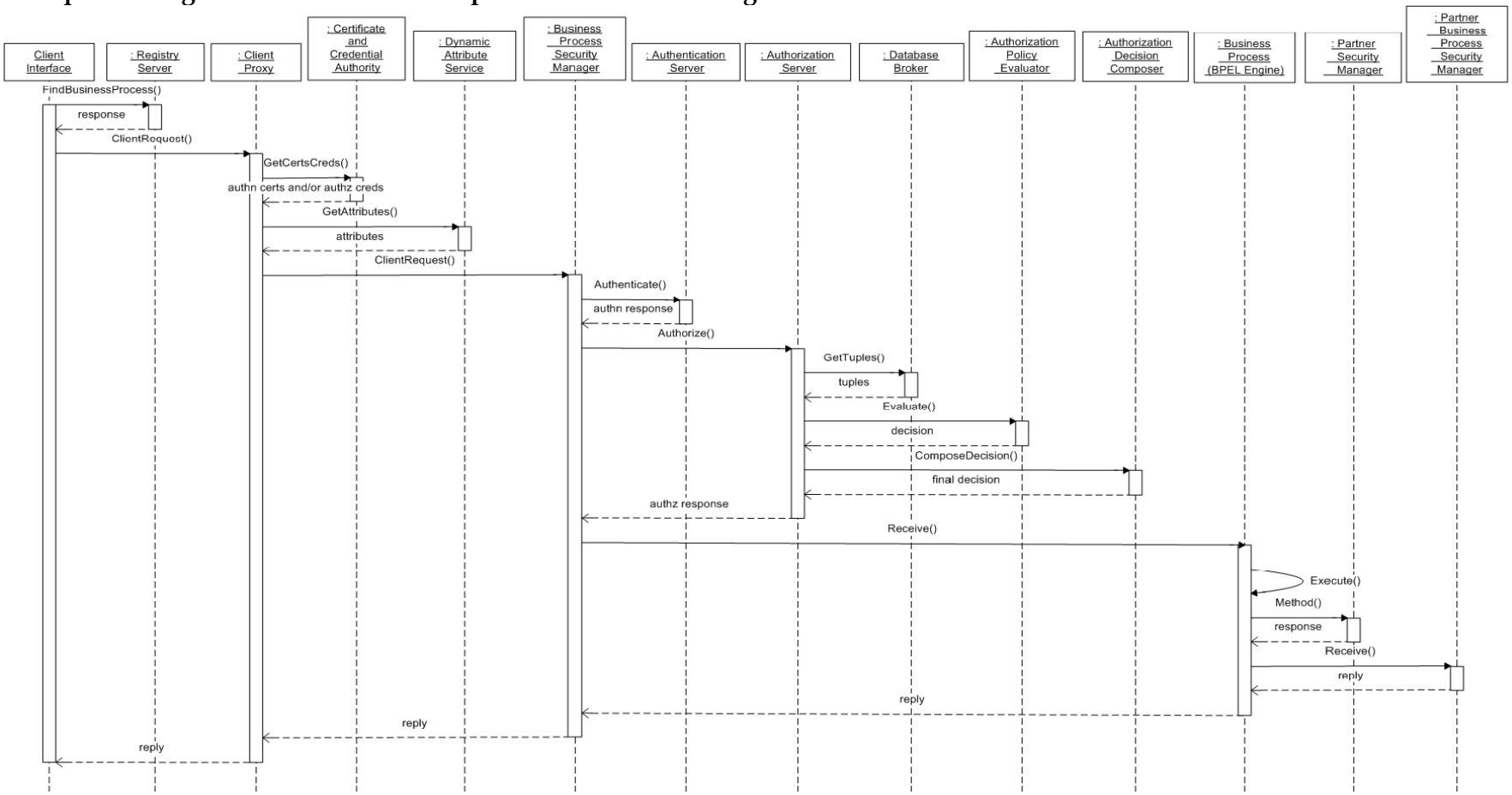

Figure A1. Push Model Authorisation Algorithm for Static Business Processes

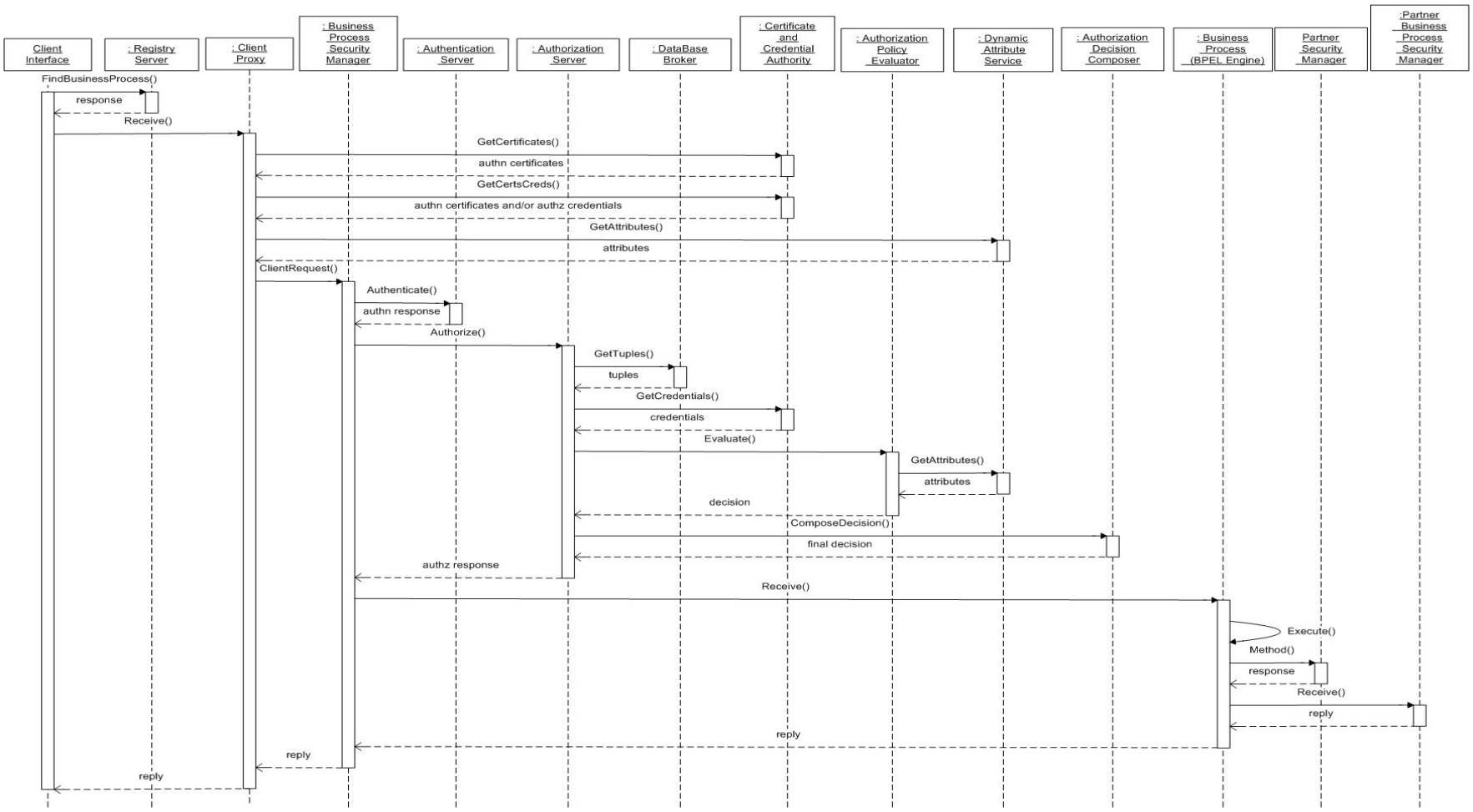

Figure A2. Pull Model Authorisation Algorithm for Static Business Processes

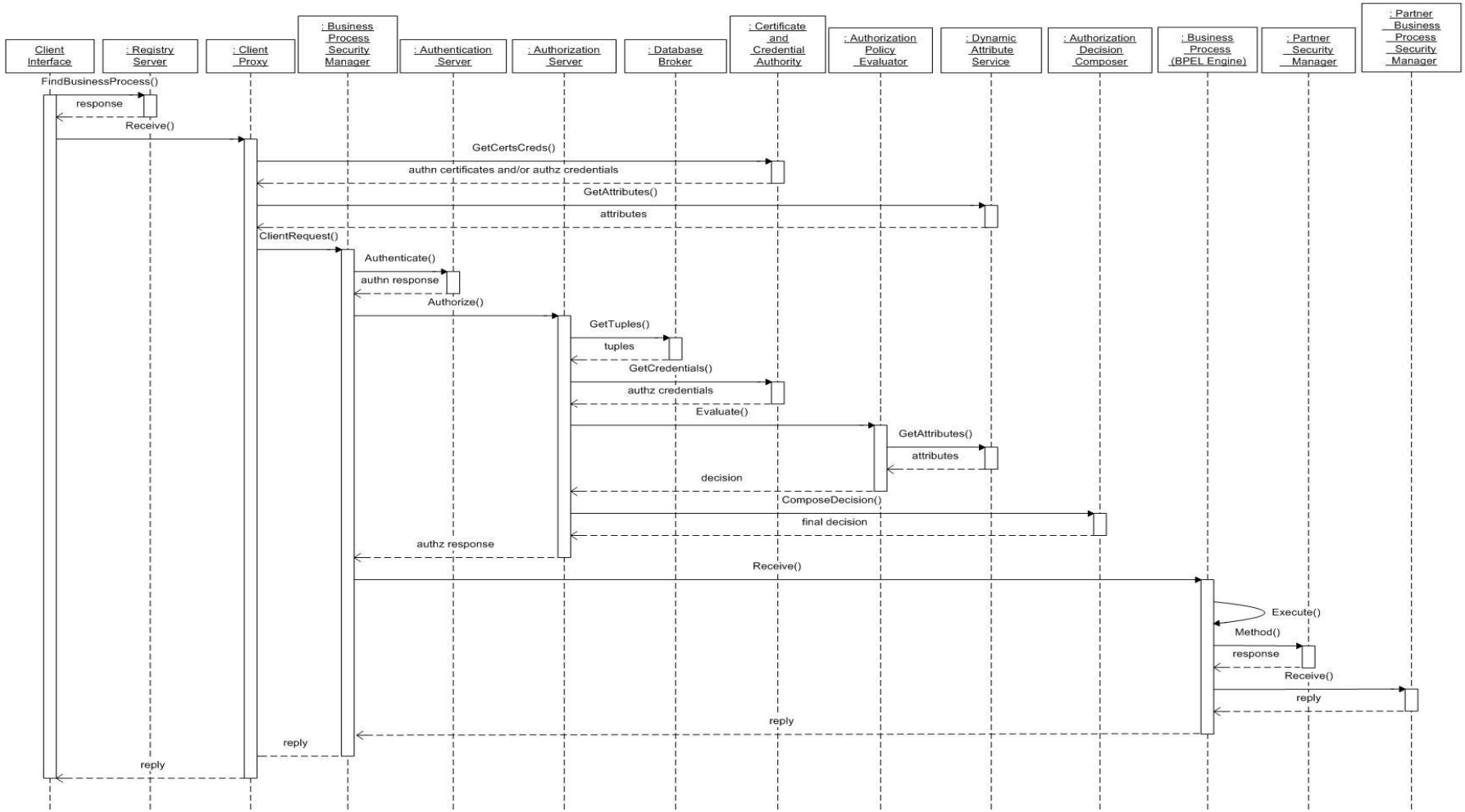

Figure A3. Combination Model Authorisation Algorithm for Static Business Processes

2. Sequence diagrams for dynamic business process authorisation algorithm

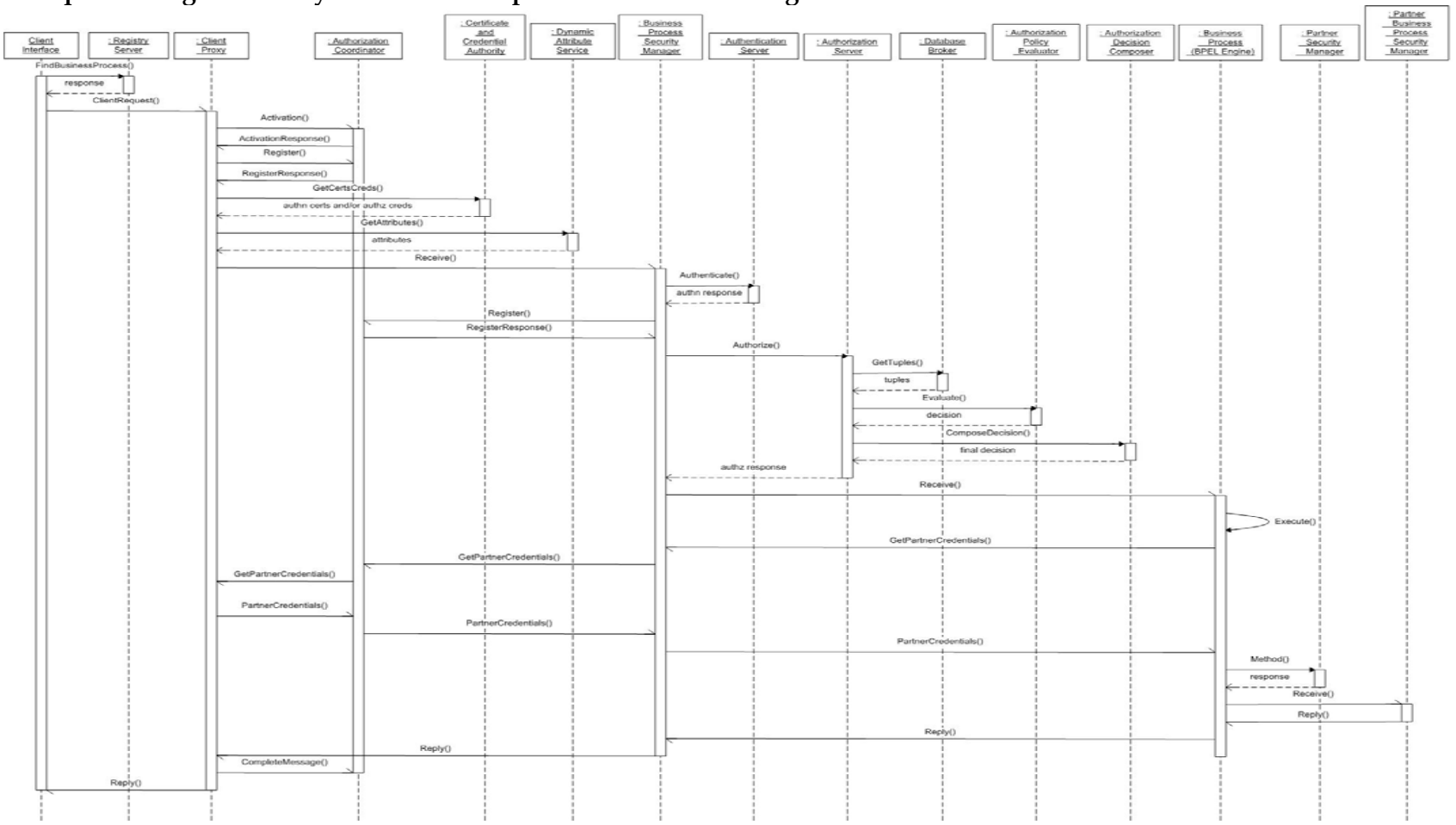

Figure A4. Push Model Authorisation Algorithm for Dynamic Business Processes

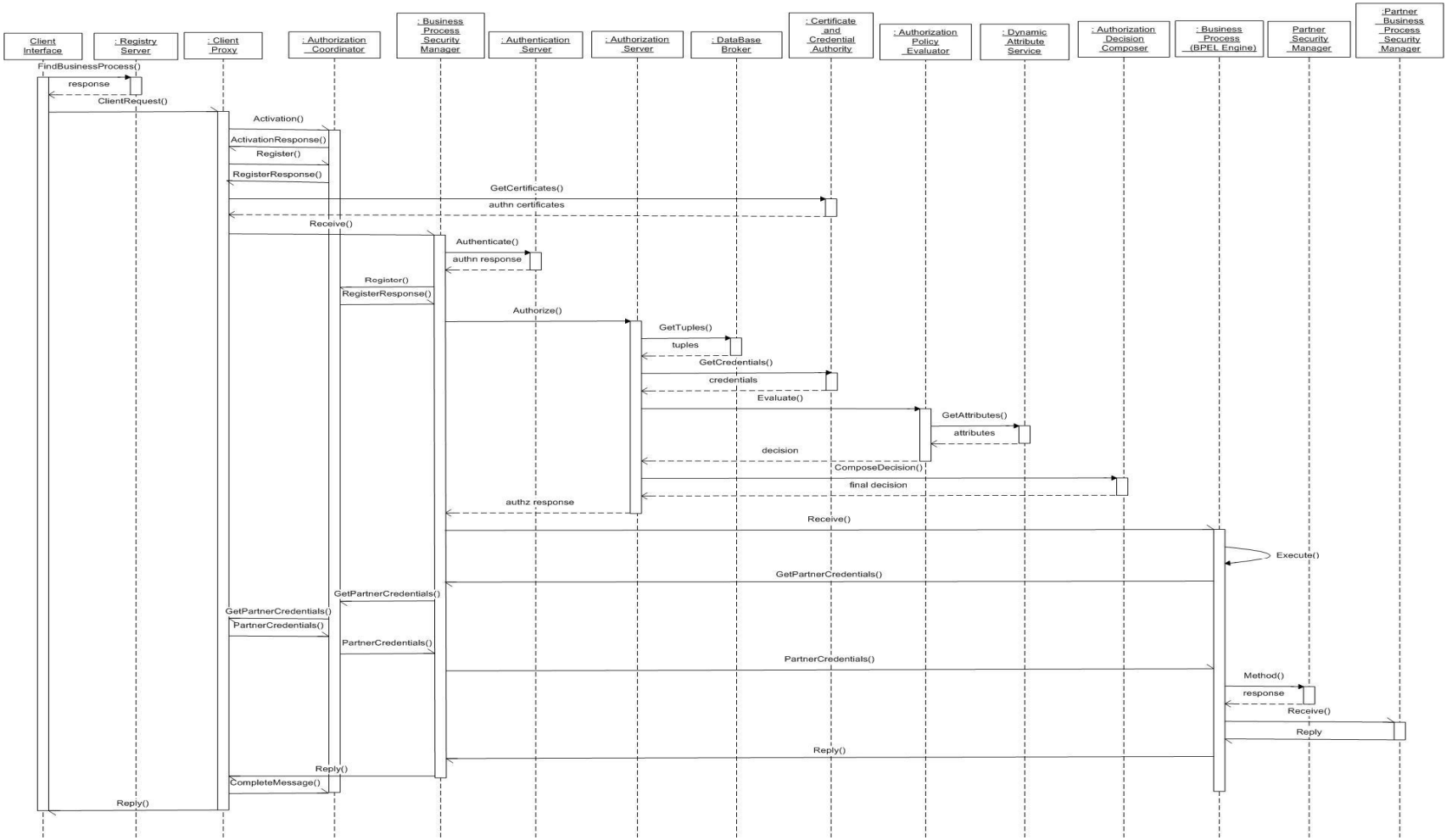

Figure A5. Pull Model Authorisation Algorithm for Dynamic Business Processes

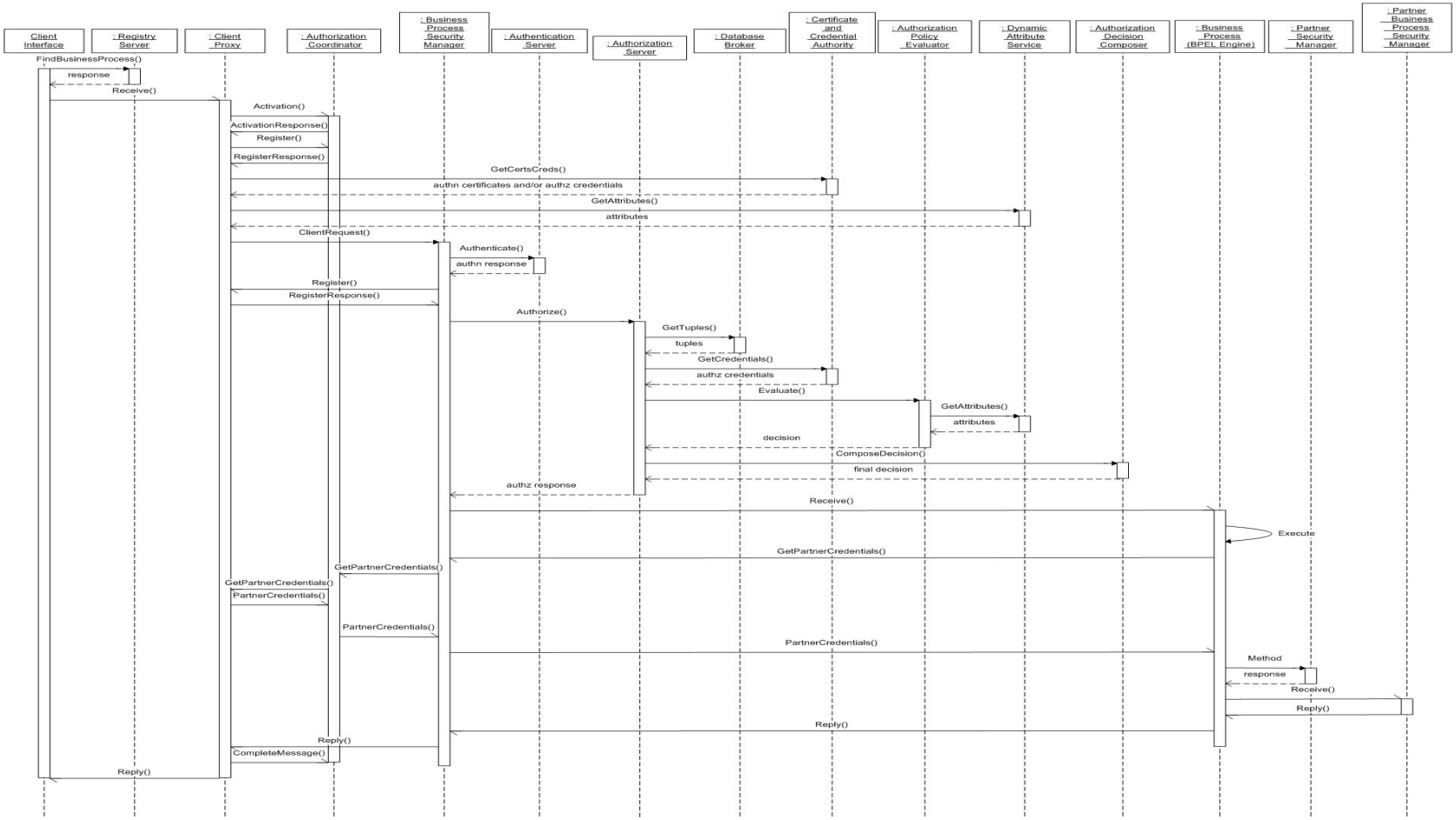

Figure A6. Combination Model Authorisation Algorithm for Dynamic Business Processes

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

On the Design of a Comprehensive Authorisation Framework for Service Oriented Architecture (SOA)

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

Sarath Indrakanti

5. CORPORATE AUTHOR

DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh South Australia 5111 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TN-1193	AR-015-670	Technical Note	July 2013

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
2012/1175751/1	07/012	Brian Palm, ASD	47	38

13. DSTO Publications Repository

http://dspace.dsto.defence.gov.au/dspace/

14. RELEASE AUTHORITY

Chief, Cyber and Electronic Warfare Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS Yes

18. DSTO RESEARCH LIBRARY THESAURUS

Web services, SOA, Business Processes, Authorisation, Access control

19. ABSTRACT

Service Oriented Architecture (SOA) has attracted considerable industry attention because of the benefits it offers such as allowing interoperability over a heterogeneous environment, amongst others. However, security is one of the main roadblocks for enterprises when it comes to the development and deployment of their SOAs. Although there are several SOA security standards available, there is as yet no standard available for SOA authorisation. In this report, we propose a comprehensive authorisation framework for SOA.